Automatic Label Placement in Maps and Figures: Models, Algorithms and Experiments

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Benjamin Niedermann

aus Überlingen

Tag der mündlichen Prüfung:	02. Februar 2017
Erster Gutachter:	PD Dr. Martin Nöllenburg
Zweiter Gutachter:	Prof. Dr.-Ing. Jan-Henrik Haunert

Danksagung

An erster Stelle möchte ich Martin Nöllenburg danken, der mich schon während meines Studiums an die Geovisualisierung herangeführt und mein Interesse daran bestärkt hat. Während meiner gesamten Promotionszeit war er durchgehend ein wichtiger und zuverlässiger Ansprechpartner in all meinen Fragen und Belangen. Selbst sein Wechsel an die TU Wien hat dem engen Betreuungsverhältnis keinen Abbruch getan, für was ich ihm sehr dankbar bin! Speziell werden mir die Forschungsaufenthalte bei ihm in Wien in besonderer Erinnerung verbleiben, die von vielen konstruktiven und bereichernden Diskussionen geprägt waren.

Besonderer Dank gilt auch Dorothea Wagner, die mich an ihrem Lehrstuhl aufgenommen hat. Die Zeit an ihrem Lehrstuhl ist für mich mit viel Freude und vielen spannenden Erlebnissen verbunden. Besonders dankbar bin ich ihr für ihr großes Vertrauen, das sie ihren Mitarbeitern schenkt. Hierdurch konnte ich viele eigene Erfahrungen sowohl in der Forschung, auf Reisen als auch in der Lehre sammeln. Diese Erfahrungen möchte ich nicht missen.

Dass die letzten viereinhalb Jahre eine besondere Zeit für mich waren, liegt natürlich auch an den großartigen Kolleginnen und Kollegen, die mich in dieser Zeit begleitet haben. Ich konnte fachlich viel von ihnen und mit ihnen lernen. Über die Arbeit hinaus sind sie aber auch zu Freunden geworden. Ohne euch wäre die Zeit nicht ansatzweise so schön geworden!

Besonders danken möchte ich Andreas Gemsa, der mir von meinem ersten Tag an mit vielen Ratschlägen und Diskussionen zu Seite stand. Allem voran die nachmittäglichen Diskussionen am Whiteboard sowie die Reise nach Hongkong werde ich nicht so schnell vergessen.

Auch Lukas Barth danke ich für seine großartige Unterstützung, der meine Projekte zuerst als Hiwi und dann als Kollege teilweise bis in die frühen Morgenstunden mittrug – und dies stets mit viel Humor.

Meiner langjährigen Bürokollegin Franziska Wegner gebührt ebenfalls großer Dank. Ich konnte mich immer auf ihre Unterstützung, Hilfe und Ratschläge verlassen. Außerdem hat sie die Zeit im Büro schlicht und ergreifend zu einer unterhaltsamen und angenehmen Zeit gemacht. Danke auch dafür, dass du stets meine durchaus verbesserungswürdigen Wortwitze mit so viel Humor ertragen hast.

Besonders erwähnen möchte ich Moritz Baum, der zum einen große Teile dieser Arbeit Korrektur gelesen hat und zum anderen in den letzten Wochen meines Aufschreibens zu einer wichtigen moralische Stütze geworden ist. Allein schon die regelmäßigen Pizzahausgänge und die vielen lustigen Gespräche haben ihren Teil dazu beigetragen!

Danken möchte ich auch Thomas Bläsius für die vielen spannenden Spieleabende mit Trauben und Käsewürfeln. Damit hat er nicht nur für abwechslungsreiche Abende gesorgt, sondern mich gleichzeitig in die faszinierende Welt der kooperativen Brettspiele eingeführt.

Obgleich ich sie nicht alle namentlich aufführe, danke ich umso mehr all meinen Koautoren für die vielen interessanten Diskussionen, ihre Unterstützung und enge Zusammenarbeit.

Schließlich bin ich auch meiner Partnerin Sonja für Ihre Geduld und Unterstützung unermesslich dankbar. Eine Promotion hat durchaus auch anstrengende und kräftezehrende Phasen. Dabei ist Sonja in all meinem Tun ein wichtiger und unersetzlicher Rückhalt für mich gewesen – dafür danke ich ihr sehr!

Deutsche Zusammenfassung

Die wachsende Menge an geographischen Daten und deren ständiger Wandel machen automatische Verfahren in der Kartografie zunehmend wichtiger. Dies betrifft insbesondere die zeitaufwendige und anspruchsvolle Aufgabe, Beschriftungen in Karten zu platzieren. Da schlecht platzierte Beschriftungen eine Karte schnell unleserlich machen, reichen einfache Verfahren nicht aus. Vielmehr verlangt die zufriedenstellende Beschriftung von Karten nach der Lösung von komplexen Optimierungsproblemen.

Die genaue Platzierung einer Beschriftung hängt stark vom Typ des Kartenmerkmals ab, das beschriftet werden soll. Für Punktmerkmale (z.B. Städte auf Karten mit kleinem Maßstab) werden Beschriftungen typischerweise dicht an das Merkmal platziert, während für Linienmerkmale (z.B. Flüsse oder Straßen) der Name entweder entlang oder innerhalb des Merkmals platziert wird. Unabhängig von der angewendeten Technik sollen Beschriftungen sich nicht überdecken, aber die Merkmale deutlich identifizieren.

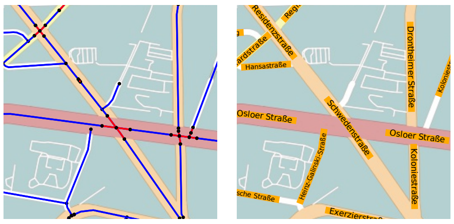
Die Problematik der Positionierung von Beschriftungen ist nicht auf Karten beschränkt, sondern tritt in vielen anderen Bereichen auf, in denen Abbildungen beschriftet werden. So ist die automatische Beschriftung von Infografiken sowie von wissenschaftlichen Zeichnungen (z.B. medizinische Zeichnungen der menschlichen Anatomie) nicht ausreichend gelöst. Stattdessen werden Beschriftungen für Abbildungen in professionellen Werken häufig noch in aufwendiger Handarbeit platziert.

Ziel der Arbeit ist der Entwurf von neuen mathematischen Modellen und Algorithmen für die Beschriftung von Landkarten und Abbildungen. Hierzu folge ich dem Paradigma des *Algorithm Engineering*, das neben der Modellierung und des Algorithmenentwurfs die Implementierung und experimentelle Evaluation der entwickelten Algorithmen in den Vordergrund der Arbeit stellt. Bereits bestehende mathematische Modelle werden verbessert und auf noch nicht betrachtete Problemvarianten erweitert. Untersuchungen zur Komplexität helfen die betrachteten Probleme einzuordnen (z.B. durch Beweis der NP-Schwere eines Problems) und entsprechende Lösungsansätze zu verfolgen (z.B. Approximationsalgorithmen bei NP-Schwere). Die entwickelten Modelle sind außerdem möglichst realitätsnah gewählt, sodass sie in der Praxis Anwendung finden können. Beim Entwurf der Algorithmen liegt der Schwerpunkt auf beweisbaren Laufzeit- und Gütegarantien. Die Anwendbarkeit der Modelle und Algorithmen wird mithilfe von Experimenten und Benutzerstudien nachgewiesen. Im Folgenden werden die erzielten Ergebnisse zusammenfassend beschrieben.

Statische Straßenbeschriftung. Während die Punktbeschriftung im Bereich der Algorithmik ausführlich untersucht wurde, gibt es für die automatische Beschriftung

von Linienmerkmalen kaum Vorarbeiten. Bisherige Arbeiten schränken entweder die Art des betrachteten Straßennetzwerks stark ein (z.B. gitterförmige Netze) oder präsentieren ausschließlich einfache Heuristiken.

In dieser Arbeit wird ein Modell für allgemeine Straßennetze vorgestellt: Aus geometrischer Sicht ist eine Straßenkarte eine Repräsentation eines *Straßengraphen* G als Arrangement von Kurven vorgegebener Breite in der Ebene. Eine Straße ist hierbei ein zusammenhängender Teilgraph von G (typischerweise ein Pfad) und jede Kante gehört zu genau einer Straße. Straßen schneiden sich in Kreuzungen, repräsentiert durch die Knoten von G. Die Straßennamen sollen so innerhalb der Kurven platziert werden, dass sie eindeutig die Straßenabschnitte zwischen den Kreuzungen benennen. In vielen Beschriftungsproblemen wird die Anzahl der Beschriftungen maximiert. Dagegen liegt der Fokus dieser Arbeit darauf, so viele Straßenabschnitte wie möglich zu benennen. Durch diese angepasste Zielfunktion wird erreicht, dass nicht unnötig viele Beschriftungen die Karte verdecken. In der Arbeit wird bewiesen, dass dieses Problem im Allgemeinen NP-Schwer ist. Für den Spezialfall, dass G ein Baum ist, wird jedoch ein Algorithmus vorgestellt, der dieses Problem in polynomieller Zeit löst.

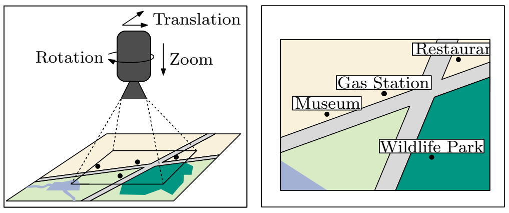


Links: Extrahierter Straßengraph G.
Rechts: Berechnete Beschriftungen.

Obwohl Straßennetze im Allgemeinen keine Baumstruktur besitzen, kann dieser Algorithmus als Grundlage für eine empirisch gut funktionierende und schnelle Heuristik verwendet werden. Hierfür wird zuerst untersucht, wie der Straßengraph G aus einem gegeben Straßennetzwerk extrahiert und vereinfacht werden kann. Des Weiteren wird anhand von Experimenten auf der Grundlage von Echtweltdaten gezeigt, dass die vorgestellte Heuristik in angemessener Zeit Ergebnisse liefert, die nahe an die entsprechende mathematisch optimale Lösung herankommen. Hierbei wird die optimale Lösung mithilfe einer geeigneten Formalisierung des Problems als ganzzahliges Programm berechnet. Trotz der NP-Schwere, können in vielen Fällen solche Programme mithilfe spezialisierter Software hinreichend schnell gelöst werden, um Vergleichswerte für eine Evaluation zu erhalten. Für eine Anwendung in der Praxis ist das Lösen ganzzahliger Programme jedoch zu langsam.

Dynamische Beschriftung von Karten. Durch die zunehmende Verbreitung von interaktiven Kartenanwendungen im Internet und auf mobilen Endgeräten eröffnen sich neue Anforderungen für die Beschriftung von Karten. Durch Grundoperationen wie Rotieren, Zoomen und Verschieben der Karte muss die Platzierung der Beschriftungen an die Änderungen der Karte angepasst werden. In diesem dynamischen Szenario reicht es nicht aus, die Anzahl der Beschriftungen zu maximieren, sondern es müssen weitere Anforderungen, sogenannte Konsistenzkriterien, erfüllt werden. So dürfen Beschriftungen nicht springen oder durch häufiges sichtbar bzw. unsichtbar werden “flackern”.

In der Arbeit wird ein allgemeines Modell für die Beschriftung von dynamischen Karten eingeführt, das die drei genannten Operationen miteinander vereint. Hierfür wird angenommen, dass der Anwender nur einen Ausschnitt der Karte sieht – als würde er die Karte mithilfe einer Kamera betrachten. Da bereits der statische Fall für das Beschriftungsproblem NP-schwer ist, ist es nicht verwunderlich, dass in dieser Arbeit auch die NP-Schwere für den betrachteten dynamischen Fall gezeigt werden kann.



Dynamische Karte. Der Anwender sieht nur einen Ausschnitt der Karte, als würde er diese mithilfe einer Kamera betrachten.

Schränkt man allerdings die Anzahl gleichzeitig angezeigter Beschriftungen ein, so kann, wie in der Arbeit gezeigt wird, das Problem mithilfe dynamischer Programmierung in polynomieller Zeit gelöst werden. Diese Einschränkung ist insbesondere für die Anwendung auf Navigationsgeräten interessant, die zumeist nur einen kleinen Bildschirm besitzen und den Anwender nicht mit zu vielen angezeigten Zusatzinformationen überfordern sollen. Dieser Algorithmus weist jedoch eine schlechte asymptotische Laufzeit auf und das Problem kann beweisbar kaum schneller optimal gelöst werden. Deshalb werden in einem weiteren Teil der Arbeit Approximationsalgorithmen sowie Heuristiken vorgestellt und experimentell evaluiert. Ähnlich zu den bereits genannten Problemstellungen, werden die Algorithmen mithilfe von ganzzahliger Programmierung bezüglich ihrer Qualität evaluiert und ihre Praxistauglichkeit gezeigt.

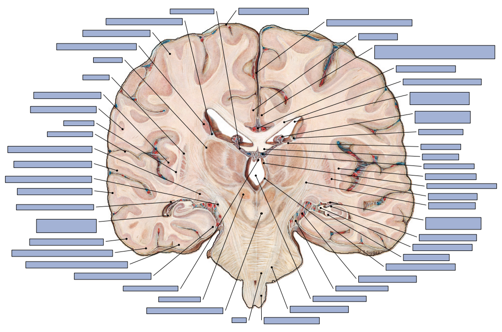
Beschriftung von Linienplänen. Linienpläne sind schematische Karten für Transportnetze wie zum Beispiel für U-Bahn- oder Straßenbahnnetze. Im Gegensatz zu Straßenkarten liegt allerdings der Fokus auf der klaren Darstellung der Netzwerktopographie und weniger auf der akkurat geographischen Widergabe des Netzwerkes. Dementsprechend umfasst das Zeichnen von Liniennetzen zwei anspruchsvolle Schritte, nämlich das Zeichnen des Netzes selbst sowie das Platzieren von überlappungsfreien Stationsnamen.

U-Bahn-Fahrplan von Wien mit berechneten Beschriftungen.

Ausgehend von einem bereits gezeichneten

Netzplan, wird in dieser Arbeit das automatische Platzieren von Stationsnamen betrachtet. Im Gegensatz zu anderen Beschriftungsproblemen dürfen Beschriftungen nicht ausgespart werden, sondern jede Station muss beschriftet werden. In der Arbeit wird ein Modell eingeführt, das sich durch seine hohe Flexibilität und Unterstützung verschiedener Zeichenstile auszeichnet. In diesem Modell ist es allerdings bereits NP-schwer eine einzelne Bahnlinie zu beschriften. Für weitere praxisnahe Einschränkungen wird jedoch ein Algorithmus vorgestellt, der das Problem für eine einzelne Bahnlinie unter Berücksichtigung einer Kostenfunktion in polynomieller Zeit optimal löst. Basierend auf diesem Algorithmus wird ein Arbeitsablauf zur Beschriftung mehrerer Bahnlinien präsentiert. Zwar können für diesen keine mathematischen Gütegarantien gegeben werden, allerdings belegt eine experimentelle Evaluation, dass die erzielten Ergebnisse nahezu die mathematisch optimale Lösung erreichen. Optimale Lösungen wurden hierfür wieder mithilfe von ganzzahliger Programmierung berechnet.

Randbeschriftung. Wissenschaftliche Zeichnungen (z.B. medizinische Zeichnungen der menschlichen Anatomie) weisen häufig eine hohe Informationsdichte auf. Um die Abbildung möglichst wenig zu überdecken, werden deshalb (im Gegensatz zu Landkarten) die Beschriftungen nicht direkt am zu beschriftenden Bildmerkmal, sondern außerhalb des Bildes platziert. Eine dünne Führungslinie zwischen Bildmerkmal und Beschriftung garantiert, dass der Betrachter eine Beschriftung ihrem Bildmerkmal korrekt zuordnen kann. Die konkrete Platzierung der Beschriftung hängt hierbei von verschiedenen Kriterien ab (z.B. relative Lage der Beschriftung, Kreuzungsfreiheit bzw. Länge der Führungslinien, usw.). Dieses Beschriftungsproblem wurde aus algorithmischer Sicht bereits ausgiebig untersucht. So wurden eine Vielzahl mathematischer Modelle eingeführt, die sich vor allem in der Art der Führungslinien unterscheiden. Neben geradlinigen Verbindungen, werden u.a. auch Führungslinien in *L*-Form, *S*-Form und mit diagonalen Streckensegmenten betrachtet.


Querschnitt des menschlichen Gehirns. Die Beschriftung wurde automatisch erstellt. Aus: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23.Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München; Kapitel 12, Abbildung 116.

Im Rahmen dieser Arbeit wurde eine Benutzerstudie durchgeführt, die klar darauf hinweist, welche Arten von Führungslinien in der Praxis aufgrund ihrer Lesbarkeit zu bevorzugen sind. Hierzu wird für die einzelnen Arten der Führungslinien die Schnellig-

keit und Genauigkeit, mit denen die Teilnehmer Punktmerkmale und Beschriftungen zuordnen, untersucht. Die Ergebnisse zeigen, dass L-förmige und geradlinige Führungslinien bezüglich der Lesbarkeit zu bevorzugen sind. Des Weiteren wird basierend auf Interviews und der Auswertung von Fragebögen die Ästhetik der Führungslinienarten diskutiert. Diesbezüglich schneiden L-förmige und Führungslinien mit diagonalen Streckensegmenten am besten ab.

Darauffolgend wird die externe Beschriftung von rechtecksförmigen Abbildungen mithilfe von L-förmigen Führungslinien untersucht. Unter Zuhilfename dynamischer Programmierung werden Algorithmen für den Fall vorgestellt, dass sich die Beschriftungen an zwei oder mehr Seiten des Bildrandes befinden. Für den zweiseitigen Fall wird außerdem angenommen, dass die Beschriftungen an angrenzenden Rändern der Abbildung liegen. Vorangegangene Arbeiten betrachteten ausschließlich gegenüberliegende Ränder. Mithilfe der vorgestellten Algorithmen kann in polynomieller Zeit entschieden werden, ob eine Abbildung so beschriftet werden kann, dass sich die Führungslinien nicht kreuzen.

Abschließend steht die automatische Beschriftung von medizinischen Zeichnungen mithilfe von geradlinigen Führungslinien im Fokus meiner Untersuchungen. Basierend auf Interviews mit Domänenexperten und der semi-automatischen Analyse von medizinischen Abbildungen werden wichtige Kriterien zur Platzierung von Beschriftungen vorgestellt. Diese dienen als Grundlage für ein formales, aber flexibles und allgemeines Beschriftungsmodell. Für diese wird ein dynamisches Programm präsentiert, das bezüglich einer gegebenen Bewertungsfunktion eine optimale Beschriftung berechnet. Mithilfe von *Algorithm Engineering* wird das polynomielle, aber asymptotisch langsame dynamische Programm praxistauglich gemacht. Eine experimentelle Evaluation zeigt, dass mit dem vorgestellten Verfahren in angemessener Zeit Beschriftungen hoher Qualität erstellt werden können.

Contents

1 Introduction

“A picture is worth a thousand words”

A widely used saying that stands for itself expressing a simple but truthful observation. It is present in all areas of everyday life and finds its application in commercial advertisements, in art, or in the daily news, just to mention a few. When it comes to passing on knowledge as it is done for example in geographic maps, textbooks, information graphics or scientific papers, a single picture may not be fully sufficient, rather it explains a small detail embedded in a larger context of other figures, texts and nomenclatures. In these applications it is crucial to guarantee that the information that is intended to be conveyed by the picture is correctly perceived by the reader without any misinterpretations. This becomes even more important when pictures are utilized to built up a common language by reflecting reality and conveying proper names of the depicted objects and facts. This common language is only justified, if anyone using it has the same understanding of what the words in the language mean and to which they exactly refer to. Hence, a picture not explaining a nomenclature correctly may lead to unwanted misunderstandings. In the best case these misunderstandings can be resolved, but in the worst case this may destroy the basis of any common discourse. This inevitably raises the question how a picture or a figure in general can be created such that it passes on knowledge without introducing ambiguities creating misleading interpretations.

One answer to this question is as simple as it is obvious: The figure is annotated with information in form of texts and symbols explaining its structure and single parts. These additional elements, which we call *labels*, describe the figure in its details and give back references to the figure’s context and the depicted content. The quality of the figure essentially depends on the placement of such labels and whether the figure accurately conveys the intended information. A good label placement relies on the ability to identify the elements of a figure clearly, while keeping the figure readable.

As an example take geographic maps. They have been used for thousands of years to describe spatial information. Among many others, geographic maps are used for navigation, the description of ownership structures, object tracking and geospatial analysis in general. All of them have in common that they require an accurate and precise map. It would be inconceivable and completely in-practicable for these applications to use mere text, but the advantages of geographic maps are obvious.

First of all, a geographic map provides an intuitive way to represent spatial data in a human-readable format and may consist of multiple types of overlaid information.

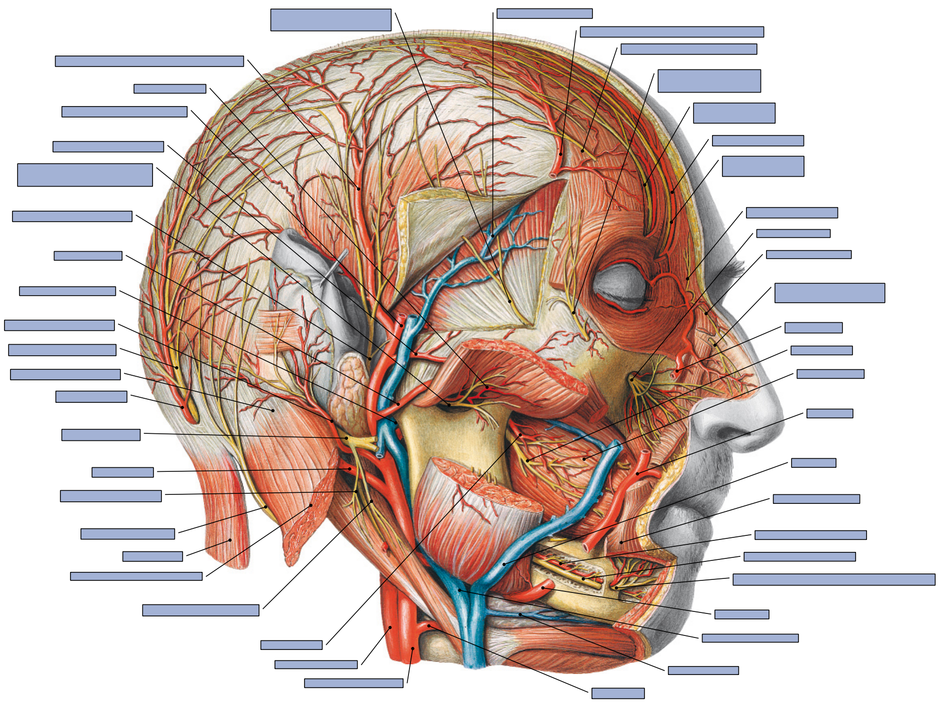


Figure 1.1: Cross section of the human head. The label placement was computed by an algorithm developed in the scope of this thesis. Source of the picture in the background: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23. Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München; Chapter 8, Figure 72.

Classically, it contains the course of streets and rivers as well as the shapes of urban areas, forest and lakes. These are only some examples, but depending on its application other spatial information can be augmented in the map. Furthermore, a geographic map typically enables the reader to easily estimate the spatial proximity as well as the relative position of objects in real world: "According to the map the house lies to the north of the lake and is closer to the lake than to the next forest". But also exact measures such as distances can be expressed in an accurate map. Despite its high informational content, the reader of the map can perceive the presented information selectively. He or she can *overlook* unimportant information while focusing on parts that actually matter for him or her – which would not be possible with mere text. However, a map without any text would be hardly useful either. Since a geographic map is a *mapping* from the real world to a schematized drawing of that world, references between both must be identified to enable the reader to actually use the map. These references are typically names identifying for example single streets, rivers, cities, lakes, woodland, regions, countries, etc. In cartography it is common to place those labels closely to their features that they name. This seemingly simple technique implements an easy-to-perceive association between labels and their respective features, if done accurately. It has become known in research as *internal label placement*. However, a readable and appealing labeling of a map cannot be taken for granted. Morrison [Mor80] estimates the time needed for labeling a map to be over 50% of the total time when creating a map by hand.

As another example take the human anatomy and its treatment in science and education. In modern medicine each single part of the human body, no matter how small, is named by applying a unified nomenclature. Needless to say that the definition of that nomenclature merely based on text would be hardly comprehensible. It is therefore not surprising that atlases of humaCn anatomy play a major role in the education of medical students and teaching the medical nomenclature. These books typically contain a myriad of filigree and detailed drawings of the human body in different cutaway views naming the single parts. For example the third volume of the popular human anatomy atlas Sobotta [PW13] contains about 1200 figures on over 384 pages. Figure 1.1 is one of these showing a cross section of the human head. Put differently, these atlases are a large collection of *maps* on the human anatomy, which give the reader a better understanding of the structure and nomenclature of the human body. In contrast to geographic maps these drawings are typically less schematized, rather they reflect the general structure of the body accurately. Further, such drawings may contain easily up to 40 or more features to be named possibly lying closely to each other. Hence, simply placing the terms defining the body parts closely to their features may yield a strongly occluded figure such that the labeling impairs the overall usefulness and appearance of the figure. Further requirements such as grouping the terms semantically may exacerbate the problem, if such a placement is possible at all. Instead the labels are placed around the figure without overlapping it and are associated by thin connecting lines to their features. This allows placing labels independently from the features' positions. This labeling technique has become known in research as *external label placement*.

Both examples—for internal and external labeling—show that label placement for figures and maps easily becomes an intricate challenge that consumes a great part of the figure's creation time. Large and dense sets of features to be labeled combined with little space yield the complex nature of the problem: Labels must clearly name their features, while keeping the remaining figure readable. Especially in cartography the label placement is complex and time consuming, which is also reflected by its computational complexity. Selecting a maximum set of disjoint labels is NP-hard, when allowing four or more possible label positions for each feature [MS91, FW91]. Even simple related sub-problems as the selection of a maximum number of disjoint rectangles (possibly modeling labels) is NP-hard [FPT81].

At the latest with upcoming dynamic maps as they are used for example by web-services, labels cannot be placed by hand anymore. Firstly, the data underlies a steady change, which also requires a steady relabeling. Secondly, dynamic maps typically provide operations such as rotation, zooming and translation. Thus, in these scenarios labels are not only placed once anymore, but for each displayed frame the selection and positioning of the labels must be done. This introduces new requirements to the label placement. For example through the sequence of displayed frames the motion of the labels should be consistent without creating distracting effects as flickering and jumping [BDY06]. It is therefore not surprising that label placement remains NP-hard in the dynamic case; e.g., see [BDY06, Bee+10]. Altogether, label placement in maps is a complex computational problem for both static and dynamic maps, and requires automatic tool-support to be handled adequately.

In other scenarios the expense is not caused by a large amount of labels, but by the number of figures to be labeled. For example labeling a single medical drawing of human anatomy is manageable in reasonable time. In interviews that were conducted in the scope of this thesis a designer stated that he needs up to two hours to create the layout of a double page of an atlas of human anatomy including the label placement. However, for one volume of such a book hundreds of medical drawings must be labeled applying the same design rules to obtain readable labelings with unified appearances. Doing the label placement by hand for a complete atlas therefore requires a lot of experience and time. Automatic tools may help the designer to reduce the time that he or she spends on labeling a figure – time that the designer can use to focus on other layout problems that need his or her expertise. Similarly to cartography, when considering interactive systems with dynamic components, handmade solutions for external labeling are not applicable anymore, rather automatic approaches become imperative.

Consequently, much research has been invested to automate the label placement in cartography as well as in figures in general. In 1972 Yoeli undertook the first algorithmic considerations on label placement in cartography. This was the beginning of a long line of scientific publications on automatic label placement and its numerous facets. The ACM Computational Geometry Task Force report [C399] particularly identifies label placement as an important research problem. For a detailed overview of the research on internal and external label placement refer to Chapter 3 and Chapter 10, respectively. This thesis builds up on the preceding research and introduces new mathematical models and algorithms for automatic internal and external label placement. The following section gives an overview of the contribution of this thesis.

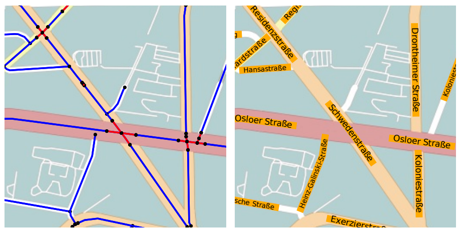
1.1 Outline and Contribution

This thesis is divided into two parts discussing internal and external label placement separately. To tackle the problems algorithmically, we apply the paradigm of *algorithm engineering*, which equally focuses on the problem modeling, the algorithm design as well as the experimental evaluation of the algorithms applied on realistic settings. Investigations assessing the computational complexity help to categorize the problems (e.g., by proving NP-hardness) and to pursue approaches adequately (e.g., approximation algorithms). Mathematical guarantees on quality and running time are at the forefront of the algorithms' design. The applicability and practicability of the developed models and algorithms are verified by experimental evaluations on real-world data as well as by user-studies.

For a short introduction to the basic concepts and techniques (e.g., NP-hardness, computational geometry, dynamic programming etc.) used throughout this thesis refer to Chapter 2.

Part I – Internal Label Placement

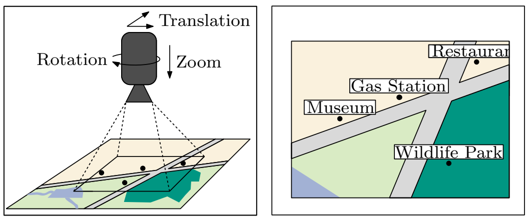
For internal label placement we focus on the application of geographic maps and present algorithms for static and dynamic maps. In Chapter 3 we first review existing models and give an overview of preceding research. Following Imhof [Imh75], we distinguish the label placement for point features (e.g., cities in small scale maps), line features (e.g., roads) and area features (e.g., lakes and woodland). We first present theoretical results (Chapter 4) as well as practical results (Chapter 5) for line features considering the application of road networks, then we switch to point features in dynamic maps again presenting theoretical (Chapter 6) and practical results (Chapter 7) separately. Finally, we consider point feature labeling for the special case of static metro maps that require that each station is labeled. We again split our results into a theoretical part (Chapter 8) and an experimental evaluation (Chapter 9).



Left: Extracted road graph G.
Right: Computed labeling.

Label Placement in Static Road Maps (Chapter 4 & Chapter 5). While label placement for point features in static maps has been extensively investigated, label placement for line features such as roads has been rarely considered. Preceding research on road labeling either strictly restricts the type of the road network (e.g., grid-shaped networks) or exclusively presents simple heuristics. In Chapter 4 we present a versatile labeling model for general road networks: Geometrically a road map is a representation of a road graph G based on an arrangement of curves with certain width. Each road is a connected subgraph of G (typically a simple path) and each edge belongs to exactly one road. Roads may intersect each other in junctions, the vertices of G. We call an edge connecting two junctions a *road section*. While in many labeling problems the number of placed labels is maximized, in this model we maximize the number of labeled road sections. By means of this adapted objective function, we achieve that no more road labels than necessary occlude the map. We prove that this optimization problem is NP-hard in general, but can be solved in polynomial time if G is a tree.

Although road networks do not form trees in general, we can use this algorithm as basis for a sophisticated heuristic, which we introduce in Chapter 5. To that end, we first present, how the road graph G can be extracted from a given road network. Based on experiments on real-world data we show that the heuristic yields near-optimal results in appropriate time. Hereby we obtain the optimal solution by means of mathematical programming.

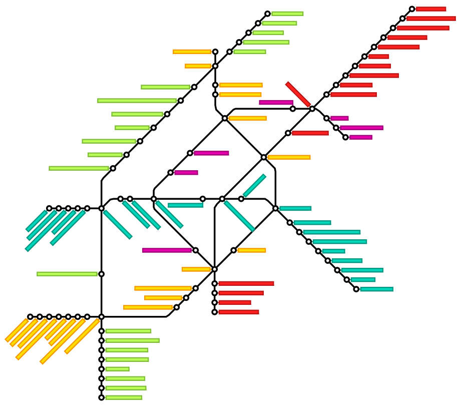


Dynamic map. The user sees only a section of the map, as if the user views the map through a camera.

Temporal Map Labeling (Chapters 6 & Chapter 7). With the upcoming of the digital age, maps became dynamic allowing various operations to change the map's perspective and its informational content. Typically, these *digital maps* offer the three basic operations *panning*, *zooming* and *rotation* to the user. These new dynamics also impact the placement of labels. While in the static scenario the number of placed labels is maximized, in the dynamic scenario this is not applicable anymore, but further requirements such as so called consistency criteria must be satisfied. Accordingly, labels may neither *jump* nor *flicker* by rapidly switching them on and off [BDY06]. In Chapter 6 we present a general model for labeling dynamic maps, which unifies the three mentioned operations. To that end, we assume that the user only sees a small part of the map – as if he or she looks through a camera shooting the map. We first investigate the model from a theoretical perspective. Since the labeling problem is already NP-hard in the static case, it is not surprising that the dynamic case is also NP-hard. We prove that it is even W[1]-hard, which implies that we cannot expect to find a fixed-parameter tractable algorithm. However, introducing some further geometric assumptions, the problem admits constant-factor approximations. Further, we consider the restricted case that at most k labels are allowed to be displayed at the same time, which is especially relevant for small screen devices such as navigation systems. In these use-cases the user should not be distracted by too many displayed labels. We show that this problem variant is solvable in polynomial time.

In Chapter 7 we switch from the theoretical considerations to a practical evaluation. To that end, we introduce some simple, but fast heuristics. Based on an experimental evaluation on real-world data, we show that these heuristics achieve near optimal solutions in appropriate time. Similarly for the road labeling problem, we have obtained the optimal solutions by mathematical programming.

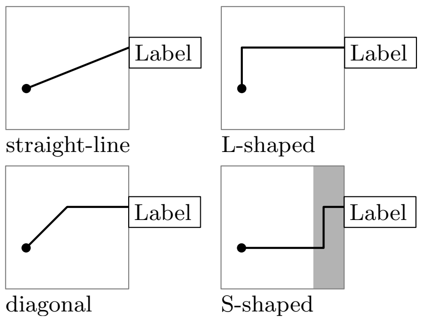
Label Placement in Metro Maps (Chapter 8 & Chapter 9). Metro maps are schematic maps that illustrate metro networks of cities. In contrast to road maps, the focus lies on the visualization of the network topology rather than on the geographic rendering of the network. Hence, drawing metro maps comprises two algorithmically complex problems, namely drawing the network itself as well as the placement of the labels.


Metro-map of Vienna. The labeling was computed by our algorithm. The layout of the map was created by the approach of Nöllenburg et al. [NW11].

In contrast to other labeling problems, labels may not be omitted, but each station must be labeled. In Chapter 8 we introduce a model that stands out by its flexibility and support of different drawing styles. However, in that model it is already NP-hard to label a single metro-line. By relaxing the problem through the introduction of some reasonable assumptions, we can present an algorithm that solves the problem in polynomial time. Based on this algorithm we present a sophisticated workflow for labeling a whole metro map in Chapter 9. For this workflow we cannot give optimality guarantees, but we show that this approach yields near optimal solutions using mathematical programming.

Part II – External Label Placement

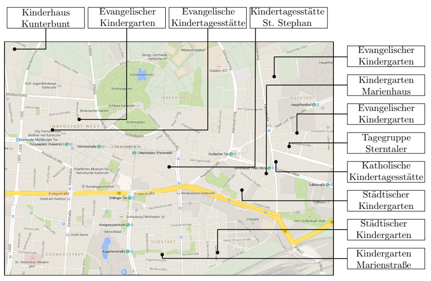
In the second part of this thesis we discuss various algorithms for external label placement. We take medical drawings in atlases of human anatomy as running example. The presented approaches for external label placement are not limited to that particular application, but can be used for other types of figures as well. In Chapter 10 we first review existing models and give an overview of previous research. In Chapter 11 we present the first user-study on the readability of different types of leaders, i.e., the lines connecting point features with their labels. In Chapter 12 and Chapter 13 we present algorithms for L-shaped leaders and straight-line leaders, respectively.



Different types of leaders.

User-Study on the Readability of Leaders (Chapter 11). Automatic external label placement has been extensively investigated both from a practical and theoretical perspective. Heuristics and exact algorithms optimizing certain objectives (e.g., minimizing the total leader length) have been proposed. While the presented heuristics mostly use straight-line leaders, the exact algorithms differ in the choice of the leader type. Typically, L-shaped and S-shaped leaders as well as leaders with diagonal segments are considered. However, these leader types have not been examined concerning their readability, yet.

In Chapter 11 we present the first formal user-study on the readability of the four most important leader types. We particularly investigate the question which of them performs best, i.e., whether and how fast a viewer can assign a feature to its label and vice versa. The results give clear indications which leader types to prefer concerning their performance, namely L-shaped and straight-line leaders. We further discuss the aesthetics of the leader types based on questionnaires and interviews conducted with the participants of the study. Concerning this matter, L-shaped leaders and leaders with diagonal segments are preferable.

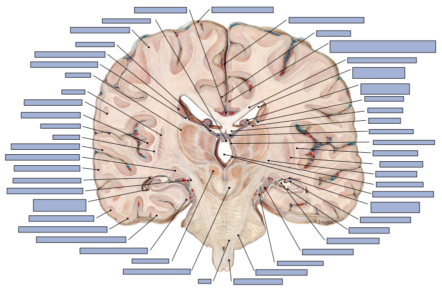

Kinderhaus Kunterbunt
Evangelischer Kindergarten
Evangelische Kindertagesstätte
Kindertagesstätte St. Stephan
Evangelischer Kindergarten
Kindergarten Marienhaus
Evangelischer Kindergarten
Tagegruppe Sterntaler
Katholische Kindertagesstätte
Städtischer Kindergarten
Städtischer Kindergarten
Kindergarten Marienstraße

A labeling with L-shaped leaders.

Multi-Sided Boundary Labeling (Chapter 12). In Chapter 12 we follow the formalization of external labeling introduced by Bekos et al. [Bek+07]. We assume that the shape of the figure is described by a rectangle R and the labels' boxes have uniform size and are already placed alongside R. Due to the uniform size of the labels, we can assume that the texts are placed inside the boxes after connecting the point features and boxes via leaders. Thus, the problem becomes a geometric matching problem connecting point features with label boxes via leaders such that the leaders do not intersect. This problem has become known as *boundary labeling*. We present polynomial-time algorithms for L-shaped leaders for the case that the labels either lie on two or more sides of R. In the two-sided case, we assume that the labels lie on adjacent sides. Considering L-shaped leaders, polynomial-time algorithms were only known for the case that the labels lie on two opposite sides. More precisely, for n labels lying on two adjacent sides we present an algorithm that checks in $O(n^2)$

time and $O(n)$ space whether a crossing free assignment between point features and labels exists. Further, we present an algorithm solving the three- and four-sided case in $O(n^4)$ and $O(n^9)$ running time, respectively.

An Algorithmic Framework for Label Placement in Figures (Chapter 13). For external label placement there are, among others, two major applications: label placement for interactive visualization systems and label placement for figures in professional books. For the former one fast approaches are required that compute labelings in real-time. Hence, fast and simple heuristics are applied accepting a loss in quality. In contrast, for the latter use-case quality is decisive, but running time plays a secondary role, which makes approaches with quality guarantees desirable. However, algorithms with provable quality guarantees have only been developed for strongly simplified boundary labeling models. It is typically assumed that the shape of the figure is a rectangle and that the labels are placed along the sides of that rectangle having uniform shapes. While these models are interesting from a theoretical perspective, they are hardly applicable, when the figure's shape is not a rectangle or the labels are not uniform.


Cross section of the human brain labeled by our algorithm. Source: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23. Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München.

In Chapter 13 we present a formal model for external labeling that is much less restrictive on the figure's shape and allows non-uniform labels. We have developed this model based on interviews with domain experts and a semi-automatic analysis of medical drawings extracted from a professional atlas on human anatomy. Using dynamic programming and straight-line leaders, we introduce an algorithm that optimally places labels along a pre-defined boundary with respect to a given cost function. This approach stands out by its generality and flexibility. It is based on few assumptions that are inherent in such labelings. Further hard constraints not to be violated as well as soft constraints to be rated by the cost function can be patched in easily. Hence, the design rules for the labelings are not a fixed component of the algorithm, but can be easily adapted by the user. This yields a flexible algorithm with quality guarantees that can be used for creating high-quality labelings. However, the strength of the approach comes at the cost of a high asymptotic running time ($O(n^8)$, where n denotes the complexity of the input). We therefore introduce further speedup techniques that make the approach practicable. In an experimental evaluation on real-world data we show that the approach produces high-quality labelings in reasonable time.

2 General Concepts and Techniques

The purpose of this chapter is the introduction of general concepts and techniques as well as a common language used throughout this thesis. Yet, the subsequent chapters are written in a self-contained style such that they are comprehensible without this chapter for a reader with experiences in computational geometry, computational complexity, and algorithm design. We start with some basic concepts on computational geometry and graphs (Section 2.1). Afterwards we give a short introduction to computational complexity (Section 2.2). In the remaining sections we discuss several general techniques used in algorithm design including approximation algorithms (Section 2.3), linear programming (Section 2.4), and dynamic programming (Section 2.5).

2.1 Basic Concepts

In this section we introduce basic concepts on (computational) geometry and graphs; more detailed introductions to these topics are found in [Ber+08, Cor+09].

Geometry. Throughout this thesis we apply Euclidean geometry. Among others, we make use of the following basic concepts; see also Figure 2.1 for an illustration.

A *point* p in the plane is an ordered pair $(x, y) \in \mathbb{R}^2$ consisting of an x-coordinate and a y-coordinate. We also write $x(p)$ for x and $y(p)$ for y. Further, we interpret a point as a 2-dimensional vector supporting the common vector operations such as addition, multiplication, Euclidean distance $\|\cdot\|$, etc. For two points p and q the *line segment* $l = \overline{pq}$ is defined as the point set $\{p + t \cdot (q - p) \mid t \in [0, 1]\}$; we say that l *starts* at p and *ends* at q. A *half-line* emanating from p through q is the set $\{p + t \cdot (q - p) \mid t \in [0, \infty)\}$. Furthermore, a *line* through p and q is the set $\{p + t \cdot (q - p) \mid t \in \mathbb{R}\}$.

A *disk* C with center c and radius r is the set $\{p \in \mathbb{R}^2 \mid \|c - p\| \leq r\}$ of all points that have at most distance r from c. The boundary ∂C of C is the set of points with exactly distance r to c; we call ∂C a *circle*. A *circular arc* is a connected segment of ∂C.

A *polygonal chain* C is a sequence $(l_1 = \overline{p_1 q_1}, \dots, l_n = \overline{p_n q_n})$ of line segments in the plane such that $q_i = p_{i+1}$ for all i with $1 \leq i < n$. We say that C is *closed* if $p_1 = q_n$ and *open* otherwise. Further, C is *simple* if only consecutive segments intersect, namely only at their endpoints. A *polygon* P is a plane region that is bounded by a closed polygonal chain C. The segments of C form the boundary of P and are called the *edges* of P. The endpoints of the segments are called the *vertices* of P. Further, P is *simple* if C is simple. The *interior* of a simple polygon P is the region that is bounded by P, while the *exterior* is the unbounded region outside of P.

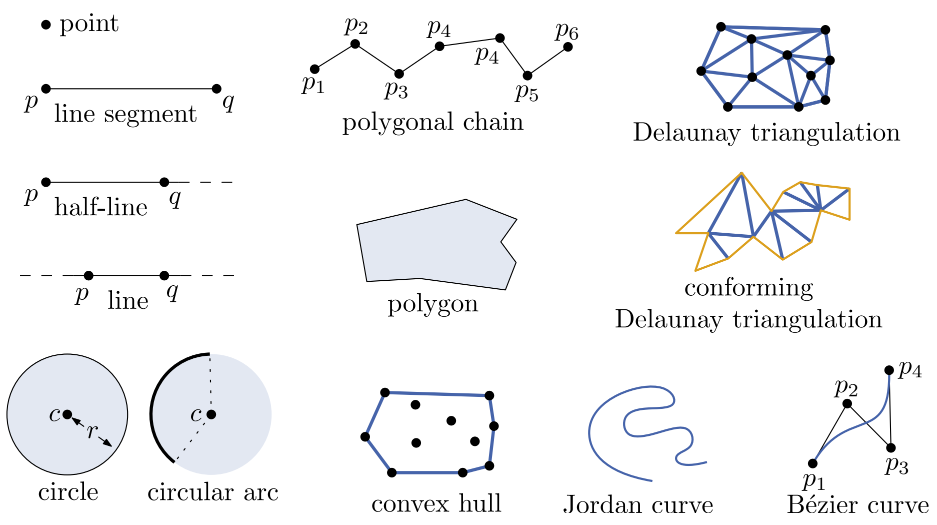


Figure 2.1: Illustration of geometric concepts used throughout this thesis.

A *quadrilateral* is a simple polygon with four edges and a *rectangle* is a quadrilateral with four right internal angles. An *axis-parallel* rectangle has two edges parallel to the x-axis and two edges parallel to the y-axis. Such a rectangle is uniquely defined by two non-adjacent vertices; we say that such two vertices *span* the rectangle.

A region $S \subseteq R^2$ is called convex, if for any two points $p, q \in S$ the line segment $\overline{pq}$ is also contained in S. A polygon P is called *convex* if its interior is a convex region. The convex hull $CH(S)$ of a set $S \subseteq \mathbb{R}^2$ is the smallest convex region containing S. If S is a finite point set, it is a well-known fact that $CH(S)$ is a convex polygon.

A *triangulation* T of a finite point set $P \subset \mathbb{R}^2$ is a maximal set of non-crossing line segments between points in P, i.e., adding any further line segment $\overline{pq}$ with $p, q \in P$ to T results in a crossing between $\overline{pq}$ and a line segment already contained in T. We observe that the line segments subdivide the plane into a set of faces. More precisely, one outer face that is the complement of $CH(P)$ and a set of inner faces that are triangles and are contained in $CH(P)$. Typically, we identify these triangles with T and say that these triangles belong to T. In this thesis we make use of *Delaunay triangulations*. A triangulation T of a point set P is called *Delaunay triangulation* if there is no point in P inside the circumcircle of any triangle of T. A Delaunay triangulation T is called *conforming* with respect to a polygon Q if T is contained in Q such that the vertices of T lie on the boundary of Q and the vertices of Q are also vertices of T.

A *curve* C in the plane $\mathbb{R}^2$ is the image of a continuous map $\varphi\colon [0,1] \to \mathbb{R}^2$. The curve C is *closed* if $\varphi(0) = \varphi(1)$ and otherwise *open*. Note that line segments, polygonal chains and circular arcs are also curves. A *Jordan curve* is a non-self-intersecting curve, i.e., φ is an injective map. Bézier curves provide a simple definition of smooth curves in the plane. In this thesis we use *cubic Bézier curves*, which are defined by four control points p_1, p_2, p_3 and p_4 as follows.

$$\varphi(t) = (1-t)^3 p_1 + 3(1-t)^2 t p_2 + 3(1-t)t^2 p_3 + t^3 p_4$$

The curve starts at p_1 and ends at p_4, while p_2 and p_3 influence the direction of the curve.

Graphs. In the following, we define basic graph notation by introducing important concepts in graph theory. A *graph* G consists of a set V of *vertices* and a set E of *edges* that relate the vertices of G to each other. More precisely, if G is *undirected* the set $E \subseteq \{\{u, v\} \mid u, v \in V\}$ consists of 2-element subsets of V, and if G is *directed* the set $E \subseteq V \times V$ consists of pairs of vertices. Hence, in the latter case the *direction* of an edge matters, while in the former case it does not. For an edge $\{u, v\} \in E$ ($(u, v) \in E$ in the directed case) we say that u and v are *adjacent*/*neighbors*. Further, $\{u, v\}$ is *incident* to u and v. A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$.

For an undirected graph $G = (V, E)$, a sequence $P = (s = v_1, \dots, v_n = t)$ of vertices in V forms a *path* if the vertices of P are pairwise distinct and there are edges $e_1, \dots, e_{n-1} \in E$ with $e_i = \{v_i, v_{i+1}\}$ for all i with $1 \le i < n$; in the directed case we require $e_i = (v_i, v_{i+1})$. We say that P *starts* at s and *ends* at t forming an s-t path. An undirected graph $G = (V, E)$ is *connected* if for each pair $u, v \in V$ there is a u-v path in G. A directed graph G is *(weakly) connected* if replacing all of its directed edges with undirected edges yields a connected undirected graph. A *connected component* of a graph G is a maximal connected subgraph of G.

Similar to paths, a sequence $C = (v_1, \dots, v_n)$ of vertices in V form a *cycle* in G if $v_1 = v_n$ and there are edges $e_1, \dots, e_{n-1} \in E$ with $e_i = \{v_i, v_{i+1}\}$ for all i with $1 \le i < n$; in the directed case we require $e_i = (v_i, v_{i+1})$. The cycle C is *simple* if all vertices of C are distinct (apart from v_1 and v_n). A graph that does not contain any cycle is *acyclic*. A graph G is a *tree* if it is connected and acyclic.

A drawing Γ of a graph $G = (V, E)$ maps each vertex $v \in V$ to a point in the plane and each edge $\{u, v\} \in E$ to a Jordan curve that starts at u and ends at v, i.e., to a non-self-intersecting continuous curve in the plane connecting u with v. A drawing Γ of G is *plane* if the Jordan curves of the edges do not cross. A *planar graph* $G = (V, E)$ admits a plane drawing. Note that such a plane drawing of G subdivides the plane into a set of faces.

2.2 Computational Complexity

The theory of computational complexity is about classifying computational problems by their difficulty. One important question is whether a problem can be solved in polynomial time or not. Here, we say that an algorithm $\mathcal{A}$ *solves* a computational problem Π, if for each instance I of Π the algorithm $\mathcal{A}$ terminates and yields the correct solution for I. Further, Π can be solved *efficiently* or in *polynomial time*, if there exists a polynomial-time algorithm $\mathcal{A}$ that solves Π, i.e., the running time of $\mathcal{A}$ lies in $O(n^c)$ for a constant c and n denoting the input size. In the following, we introduce concepts that help us to investigate the question whether a computational problem can be solved efficiently. For a more detailed introduction to computational complexity see [GJ79, Cor+09].

Computational complexity theory distinguishes, among others, two types of computational problems, namely *decision problems* and *optimization problems*. In many cases, studying optimization problems is more interesting from a practical and applied point of view, but decision problems may help us to assess the computational complexity of optimization problems. In the following, we explain this more specifically.

A *decision problem* Π is a formalized question that allows exactly two answers, namely *yes* or *no*. If for a problem instance $I \in \Pi$ the answer on the question is *yes*, we say that I is a *yes*-instance and otherwise a *no*-instance of Π.

Take the computation of an independent set of rectangles among a set of given rectangles as an example. A set I of rectangles is called an *independent set* if the rectangles in I are pairwise disjoint. For a set R of rectangles we then want to find an independent subset of R. The decision problem is formalized as follows.

Problem 2.1 (IndependentRectanglesDecision)**.**
Given: *Set R of rectangles in the plane and parameter $K \in \mathbb{N}$.*
Question: *Is there an independent set $I \subseteq R$ with $|I| \geq K$?*

We also can ask for the *best* among all feasible solutions. In that case we obtain the corresponding *optimization problem*. In our example the optimization problem is formalized as follows.

Problem 2.2 (IndependentRectangles)**.**
Given: *Set R of rectangles in the plane.*
Find: *Maximum independent set $I \subseteq R$, i.e., there is no independent set $I' \subseteq R$ with $|I| < |I'|$.*

From the perspective of computational complexity a decision problem is not *harder* than its corresponding optimization problem in the following sense: Given an algorithm solving the optimization problem, we can easily utilize it to solve the decision problem. For example assume that we are given a polynomial-time algorithm $\mathcal{A}$ solving IndependentRectangles. For a given instance (R, K) of IndependentRectanglesDecision we compute the maximum independent set $I \subseteq R$ using $\mathcal{A}$. If $|I| \geq K$, we return that the instance is a yes-instance and otherwise a no-instance. Using such a reduction in general proves that if an optimization problem is solvable in polynomial time, then also its decision problem. Conversely, if we can show that a decision problem cannot be solved in polynomial time, then its corresponding optimization problem can also not be solved efficiently, which yields an interesting negative result on the complexity of the optimization problem. This directly leads us to the following two complexity classes.

The Classes P and NP. The class P contains all decision problems that can be solved in polynomial time, i.e., for each problem in P there is a deterministic Turing machine that solves the problem in polynomial time.

In contrast, the class NP contains any decision problem for which there is a non-deterministic Turing machine that solves the problem in polynomial time. One can imagine the behavior of such a machine as follows. First, in a non-deterministic phase, the machine computes a solution for the given problem in polynomial time. Then, in a deterministic phase, it checks in polynomial time whether the solution is correct. In our example of independent rectangles such a machine first *guesses* an independent set I of rectangles in R. Since such a set has polynomial size with respect to R, the machine can do this in polynomial time. Then, in the second phase, it checks whether the rectangles are pairwise disjoint and $|I| \geq K$, which it can do in polynomial time by counting the rectangles in I and by doing pairwise intersection tests.

Obviously, it holds P⊆NP, but it is not known whether NP⊆P, which is one of the big open questions in computer science. The importance of this question relies on its far-reaching consequences. The concept of NP-hardness provides valuable insights into these consequences. We now explain this concept in detail. A *polynomial-time reduction* of a decision problem Π_1 to a decision problem Π_2 is a polynomial-time algorithm that transforms each instance I_1 of Π_1 into an instance I_2 of Π_2 such that I_1 is a *yes*-instance if and only if I_2 is a *yes*-instance. If such a reduction exists, we write $\Pi_1 \leq_P \Pi_2$. Thus, if we can solve the problem Π_2 in polynomial time, then we also can solve Π_1 in polynomial time by reducing Π_1 on Π_2.

We say that a decision problem Π is *NP-hard* if any problem Π' in NP can be reduced to Π in polynomial time, i.e., $\Pi' \leq_P \Pi$. Further, if Π also lies in NP, the problem is called *NP-complete*. Since a decision problem Π_D is not harder than its optimization problem Π_O, we say that Π_O is NP-hard if Π_D is NP-hard.

Unless P=NP, there is no polynomial-time algorithm for any NP-hard problem. Conversely, if P=NP, all problems in NP are solvable in polynomial time. Hence, proving that a problem is NP-hard provides an interesting and important classification of the problem. For example the problem IndependentRectangles is NP-hard, even if the rectangles are axis-aligned unit squares [FPT81]. In this thesis we prove NP-hardness for several labeling problems. To that end, we prove for each of these problems that there is a polynomial-time reduction from a proven NP-hard problem to the according decision variant of the labeling problem.

The Class FPT. Another complexity class is FPT. Before we formally define this class, we start with a simple example. A *vertex cover* of a graph $G = (V, E)$ is a set $S \subseteq V$ such that each edge in E is incident to at least one vertex $v \in S$.

Problem 2.3 (VertexCover).
Given: *A graph* $G = (V, E)$.
Find: *Minimum vertex cover* S *of* G*, i.e., there exists no vertex cover* S' *of* G *with* $|S'| < |S|$.

Although VertexCover is an NP-hard problem [GJ79], we can solve VertexCover such that the running time is exponential *only* in the size k of the constructed vertex cover, but polynomial in the input size of G: By enumerating all subsets $S' \subseteq V$ of size k, we can easily find a vertex cover of size k, if it exists. The running time of this algorithm is $O(2^k \cdot (|V| + |E|))$. Thus, if k is small, the running time is acceptable even for large input instances.

We generalize these observations as follows. For a parameter k a problem Π is *fixed-parameter tractable* if any instance $I \in \Pi$ with size n can be solved in time $O(f(k) \cdot p(n))$, where f is some computable function depending on k and p is a polynomial depending on n. The class of all fixed-parameter tractable problems is called *FPT*. Hence, investigating whether a NP-hard problem Π is fixed-parameter tractable can be a worthwhile approach to tackle its NP-hardness. Further, the class FPT is the base of the so called W-hierarchy, which is a family of complexity classes W[i] based on *parameterized reductions*; it holds FPT=W[0] and W[i]⊆W[j] for all $i < j$. It is unknown whether FPT=W[1], but an answer on this question would also solve the P≠NP question. Similarly to the NP-class, the concept of W[i]-hardness has been introduced to describe the *hardest* problems of the class W[i]. In particular, a W[1]-hard problem is not in FPT unless NP=P. For a detailed introduction to W-hierarchy see for example [FG06].

2.3 Approximation Algorithms

To find solutions for NP-hard problems efficiently, one often relaxes the demand of finding optimal solutions. Indeed, in many cases one can mathematically prove approximation guarantees for polynomial-time algorithms. For example, we can approximate the optimal solution of VertexCover by a factor of 2. Starting with an empty set S, consider each edge e of the given graph iteratively. If e is incident to some vertex of S then skip it and otherwise add the incident vertices of e to S. Obviously, at least one of both vertices must belong to any optimal solution. Hence, the set S is at most twice as large as an optimal vertex cover.

This is formalized as follows. Let Π be an optimization problem, and let $\mathrm{OPT}(I)$ denote the value of an optimal solution of a given problem instance $I \in \Pi$. Without loss of generality, let Π be a minimization problem; for maximization problems we can define the concepts analogously. An algorithm $\mathcal{A}$ is called a *k-approximation algorithm* for Π if $\mathcal{A}(I)/\mathrm{OPT}(I) \leq k$ for any $I \in \Pi$, where $\mathcal{A}(I)$ denotes the value of the solution computed by $\mathcal{A}$ on I. If not stated otherwise, we assume that the running time of $\mathcal{A}$ is polynomial in the input size. A *polynomial-time approximation scheme (PTAS)* for a minimization problem Π is a $(1+\varepsilon)$-approximation algorithm $\mathcal{A}$ for Π and any $\varepsilon > 0$. It is required that the running time of $\mathcal{A}$ is polynomial in the input size for every fixed ε. The algorithm $\mathcal{A}$ further is a *fully polynomial-time approximation scheme* (FPTAS) if its running time is also polynomial in $1/\epsilon$.

2.4 Linear Programming

Linear programming is a general technique to formalize optimization problems that consist of a linear objective function subject to linear constraints; see [Cor+09] for a detailed introduction. More precisely, for n variables $x_1, \ldots, x_n \in \mathbb{R}_0^+$ we are given a linear function

$$f(x_1, \ldots, x_n) = c_1x_1 + \cdots + c_nx_n$$

and a set of k linear constraints

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + \cdots + a_{2n}x_n &\leq b_2 \\ &\cdots \\ a_{k1}x_1 + \cdots + a_{kn}x_n &\leq b_2, \end{aligned}$$

where $c_i, a_{ij}, b_j \in \mathbb{R}$ are constants for $1 \leq i \leq n$ and $1 \leq j \leq k$. We then aim at finding an assignment for the variables $x_1, \ldots, x_n$ such that the constraints are satisfied and $f(x_1, \ldots, x_n)$ is maximized (or alternatively minimized).

The production of goods is a classical example for the application of linear programming. Assume that a bakery wants to produce two types B_1 and B_2 of breads. To produce a box of bread B_1, one needs 10 units of ingredient I_1, 30 units of ingredient I_2, and 20 units of ingredient I_3. Similarly, to produce a box of bread B_2 one needs 40 units of ingredient I_1, 10 units of ingredient I_2 and 20 units of ingredient I_3. The warehouse stores 600 units of I_1, 550 units of I_2 and 300 units of I_3. Further, the bakery sells a box of bread B_1 for 30\$ and a box of bread B_2 for 60\$. The bakery wants to maximize its profit.

We express the problem as the following linear programming formulation. We introduce the variables $x_1 \geq 0$ and $x_2 \geq 0$, which we interpret such that x_1 is the number of produced boxes of bread B_1 and x_2 is the number of produced boxes of bread B_2. We then aim at maximizing the profit

$$f(x_1, x_2) = 30x_1 + 60x_2$$

subject to the constraints

$$\begin{aligned}10x_1 + 40x_2 &\leq 600\\ 30x_1 + 10x_2 &\leq 550\\ 20x_1 + 20x_2 &\leq 300.\end{aligned}$$

The constraints enforce that we do not exceed the available amount of ingredients I_1, I_2 and I_3, respectively.

In 1947 George Danzig developed the so called *simplex method* to solve such formulations. Although in theory it has exponential running time in the worst case, it turns out to be a fast procedure in practice. In contrast, for the ellipsoid method introduced by Khachiyan in 1979 and the interior point method introduced by Karmarkar in 1984, it can be proven that they solve a linear programming formulation in polynomial time. For a more detailed overview on the development of linear programming approaches see for example [Cor+09].

An important special case of linear programming is the restriction of the variables to integers,[1] i.e., $x_1, \ldots, x_n \in \mathbb{N}$; we then call the technique *integer linear programming*. When only a subset of the variables must be assigned integers, the problem is called *mixed integer linear programming*. Integer linear programming formulations provide a general tool to formalize a broad spectrum of optimization problems. Solving an integer linear programming formulation is NP-hard in general [GJ79], but it turns out that in practice we can often apply specialized solvers to find optimal solutions for reasonably sized instances in acceptable time. This approach helps us to obtain optimal solutions for NP-hard problems, which we then can compare against solutions produced by typically faster heuristics.

2.5 Dynamic Programming

One of the most famous approaches in computer science is certainly the *divide and conquer principle*. An instance of a given computational problem is recursively divided into smaller components until the remaining components are small enough to be solved trivially. Ascending in the recursion tree, the obtained solutions are then assembled to finally obtain a solution for the input instance. Algorithms for sorting numbers, e.g., merge sort, are typical examples using this principle; for a detailed introduction see for example Cormen et al. [Cor+09]. If the recursion tree is exponential in the input size, however, this approach becomes unfeasible. For some of the computational problems, dynamic programming provides a way out by appropriately storing and reusing already computed results.

[1]For our example, this implies that only complete boxes of bread are produced.

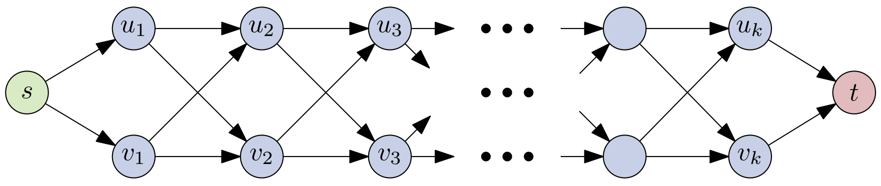


Figure 2.2: A graph that has an exponential number of s-t paths. Each s-t path either contains u_i or v_i for all $1 \leq i \leq k$. Hence, there are 2^k possibilities to assemble an s-t path.

Take the computation of the shortest path in a directed acyclic graph as an example, see also [Cor+09]. For a graph $G = (V, E)$ with cost function $c\colon E \longrightarrow \mathbb{R}$, let P be an s-t path for two vertices $s, t \in V$. We define its cost $c(P) = \sum_{e \in P} c(e)$, where we assume that P contains the edges of the path (instead of its vertices as originally defined). The path P is a *shortest* s-t path in G if there is no other s-t path P' with $c(P') < c(P)$.

Problem 2.4 (ShortestPath).

Given: *Directed, acyclic graph $G = (V, E)$ with cost function $c\colon E \longrightarrow \mathbb{R}$ and two vertices $s, t \in V$.*

Find: *Shortest s-t path P in G.*

Let P be a shortest s-t path and let P' be a subpath of P connecting the two vertices u and v. One can easily prove that P' is again a shortest path connecting u and v. Put differently, any subpath of a shortest path is again a shortest path.

A shortest path between two vertices s and t can be computed using the divide and conquer principle as follows. Assume that a shortest path $P_{s,t}$ between s and t exists and let $u_1, \dots, u_k$ be the vertices in V with $(u_i, t) \in E$ $(1 \leq i \leq k)$. Further, let P_{s,u_i} be a shortest path from s to u_i for $1 \leq i \leq k$. Obviously, we have $c(P_{s,u_i}) + c(u_i, t) = c(P_{s,t})$ for some i with $1 \leq i \leq k$. Given the paths $P_{s,u_1}, \dots, P_{s,u_k}$, we can construct $P_{s,t}$ by computing $i = \operatorname{argmin}_{1 \leq i \leq k} c(P_{s,u_i}) + c(u_i, t)$ and setting $P_{s,t} = P_{s,u_i} + (u_i, t)$, where $P_{s,u_i} + (u_i, t)$ denotes the concatenation of P_{s,u_i} with the edge (u_i, t). Applying the approach recursively, we can compute $P_{s,u_1}, \dots, P_{s,u_k}$ in the same manner. The leaves of the recursion tree correspond with single edges in G, which can be handled trivially. Thus, we can compute a shortest path between s and t using the divide and conquer principle. However, the according recursion tree becomes easily exponential in the input size; see Figure 2.2 for an example. The problem is that we compute the same shortest paths multiple times instead of reusing already computed information.

Dynamic programming provides a better solution. Instead of recomputing shortest paths again and again, we store the cost of an already computed shortest path in a table. More precisely, we create a one-dimensional table T that contains for each vertex $v \in V$ the cost of a shortest path from s to v; if v is not reachable from s, we define $T[v] = \infty$. We can recursively express $T[v]$ by

$$T[v] = \begin{cases} \min_{(u,v)\in E} T[u] + c(u,v) & v \text{ has incoming edges.} \\ \infty & \text{otherwise.} \end{cases}$$

The table entry $T[t]$ then contains the cost for the shortest path from s to t; if such a path does not exist, we have $T[t] = \infty$. A simple analysis yields that we need $O(|V| + |E|)$ time for computing all table entries: there are $O(|V|)$ table entries and we consider each edge only once. By storing the costs of already considered paths, we consequently reduce the exponential running time to a running time that is linear in the number of the graph's vertices and edges.

Typically, we are not only interested in the value of the solution, but also in the solution itself. To construct the solution from T, we can apply a standard backtracking approach. In our example, we store for each vertex v its *predecessor* $S[v] \in V$ with

$$(S[v], v) = \operatorname{argmin}_{(u,v)\in E} T[u] + c(u,v).$$

If such a predecessor does not exist, we set $S[v] = \perp$. If $T[t] < \infty$, we can construct a shortest s-t path by following the predecessors of t to s. Since we can do this in $O(|V|)$ time, we obtain $O(|V| + |E|)$ running time in total for computing a shortest s-t path in G. We will use this approach in Chapter 8 as a sub-routine, which we call MinPath.

So that dynamic programming works, it is crucial that the considered sub-instances of the problem can be described by a constant number of parameters. This allows us to create dynamic programming tables that have a constant number of dimensions, which significantly influences the asymptotic running time. Since we typically compute each table entry, the dimensions often contribute to the exponent of the running time. In our example, a problem instance is uniquely defined by two vertices s and t; any shortest path between s and t has the same cost. Since we fix s and only vary the destination vertex, it suffices to consider a table with one dimension. In this thesis we often consider dynamic programming in the context of geometric problems. Thus, the problem instances are typically described by geometric properties. By the previous reasoning it is therefore important to describe the problem instances by as few as possible geometric properties. For example, in Chapter 13 we describe a problem instance by a simple polygon. However, we do not describe the polygon by its vertices, but by at most four control points, which allows us to apply a dynamic programming approach having polynomial running time.

Altogether, dynamic programming is a generic and powerful approach that allows the systematic and efficient exploration of the solution space of many optimization problems. This is achieved by decomposing problem instances into smaller, independent sub-instances that can be described by a constant number of parameters. The running time of a dynamic programming approach essentially relies on this number.

Part I

Internal Label Placement

3 Introduction to Internal Label Placement

Internal label placement is a frequently used labeling technique in cartography to augment maps with additional information allowing that the actual content of the map is occluded by the placed labels. Eduard Imhof, a well-known cartographer and former president of the International Cartographic Association (ICA), condensed the importance of label placement as follows.

> *"Poor, sloppy, amateurish type placement is irresponsible; it spoils even the best image and impedes reading"*, Eduard Imhof (1895–1986), [Imh75]

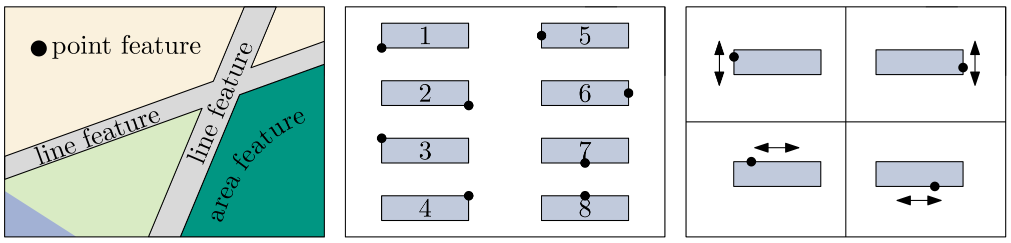
In [Imh75], which is a translation of his original work [Imh62], Imhof summarizes his experiences as cartographer and gives a detailed overview of reasonable criteria for label placement. For that purpose, he distinguishes the placement of labels for *point features*, (e.g., points of interests, cities on small scale maps, etc.), *line features* (e.g., rivers, roads, tracks, etc.) and *area features* (e.g, lakes, buildings, forests, etc.); see Figure 3.1(a). He requires that for point features the label is placed closely to its feature, for line features the label is placed along its feature, and for area features the label is placed inside its feature. Among others, he introduces three general principles:

(1) "The names should, in spite of their incorporation into the dense graphics of the map, be easily read, easily discriminated, and easily and quickly located."
(2) "The name and the object to which it belongs should be easily recognized."
(3) "Names should disturb other map contents as little as possible. Avoid covering, overlapping, and concealment."

In our developed models, we particularly pay attention to Criterion (3), requiring that labels may not overlap, because this immediately destroys their legibility. This also ensures Criterion (1) in certain extents. Criterion (2) is enforced by the considered models, which typically place labels closely to their features. In the following, we give an introduction to label placement for both static and dynamic map labeling.

3.1 Static Map Labeling

Starting with the algorithmic considerations by Yoeli [Yoe72], plenty of research efforts have been made to automate the process of label placement in cartography. First, research focused on traditional *static maps* that do not change once they are drawn. Labeling point features received most attention, while the other two feature types were considered less. Next, we present the feature types separately.



(a) Feature types. **(b)** Fixed-position model. **(c)** Slider model.

Figure 3.1: Illustrations for static map labeling. (a) The different map features proposed by Imhof [Imh75]. (b) The fixed-position model with priorities proposed by Yoeli [Yoe72]. (c) The slider model proposed by van Kreveld et al. [KSW99].

Point Feature Labeling. Yoeli [Yoe72] introduced the first formal model for labeling point features. Representing a label as an axis-aligned rectangle, he requires that a label may only hold one of eight possible positions, each having a unique priority; see Figure 3.1(b). More precisely, the label must be placed such that either one of its corners or one of its edges' midpoints coincides with its point feature. This model has become known as the *fixed-position* model for point feature labeling. Similar to Hirsch [Hir82], van Kreveld et al. [KSW99] relax this requirement and allow the boundary of the label to *slide* along the point feature, i.e., the label must be placed such that its point features lies on its boundary; see Figure 3.1(c). They further show that sliding labels allow more point features to be labeled in practice. These models have become known as *slider models* for point feature labeling.

Since none of these models guarantees that all labels can be placed without any overlap, typical optimization criteria are maximizing the number of placed labels without overlaps (label number maximization), maximizing the size of the labels without overlaps (label size maximization) and minimizing the number of labels that overlap. In the two latter cases all point features must be labeled.

Allowing only one position for each label in the fixed-position model, the label number maximization problem directly corresponds to finding a maximum independent set for a given set S of rectangles, i.e., finding a maximum number of rectangles in S such that they are pairwise disjoint. Fowler et al. [FPT81] prove that this problem is NP-hard, even if the rectangles are uniform squares. Marx [Mar05] shows that it is even W[1]-hard for uniform squares. The problem remains NP-hard if four or more positions [MS91, FW91] are allowed or if the point feature is allowed to slide along all four sides of the label in the slider model [KSW99]. Hence, theoretical research turned towards approximation algorithms. Hochbaum and Maass [HM85] propose the first polynomial-time approximation scheme (PTAS) for selecting a maximum number of disjoint unit-squares introducing the so-called *shifting technique*. Agarwal et al. [AKS98] apply the same technique to enhance the result to unit-height rectangles also obtaining a better asymptotic running time. Concerning running time, this PTAS is further improved by Chan [Cha04]. Van Kreveld et al. [KSW99] apply the shifting technique to obtain a PTAS for the same setting using the slider model. For the weighted case both the fixed position model [Erl+10] as well as the slider model [Poo+04] admit a PTAS. Furthermore, for arbitrary rectangles and the fixed position model the best known polynomial-time approximation algorithm has ratio $O(1/\log\log n)$ [CC09]. For the weighted case Adamaszek and Wiese [AW13] give an approximation scheme with quasi-polynomial running time, i.e., its running time is in $2^{O((\log n)^\epsilon)}$ for some fixed $\epsilon > 0$.

The label size maximization problem received less attention. Formann and Wagner [FW91] prove that the label size maximization problem is NP-hard for four positions. It does not even admit a polynomial-time $(2-\epsilon)$-approximation for $\epsilon > 0$ (unless NP=P), but they present an efficient 2-approximation algorithm for that case. Wagner and Wolff [WW97] introduce another 2-approximation algorithm that yields better results in practice. Qin and Zhu [QZ02] give a 2-approximation algorithm for the slider model and Jiang et al. [Jia+05] present a constant-factor approximation algorithm for circular labels.

Inspired by air-traffic control, de Berg and Gerrits [BG12] consider the problem that neither labels may be left out nor their sizes may be changed, but applying the common fixed-position and slider models, they aim at maximizing the number of labels that do not overlap. Considering labels as unit-squares, they present efficient constant factor approximation algorithms and prove the existence of a PTAS.

Besides these approximation algorithms, further approaches are used to tackle the problem of label placement. Among others, integer linear programming approaches [Zor86] as well as rule-based systems mimicking the placement by cartographers [FA84] are suggested. Further, label placement by means of simulating annealing [CMS94, CMS95] as well as force-directed label placement procedures [EKW03] have been subject to research. We have mentioned the earliest representatives of these techniques, but there is much more research that builds up on those results. For a more detailed overview see [KB08].

Line Feature Labeling. Imhof [Imh75] requires in his cartographic criteria that names of line features are placed alongside the line features, but "complicated and extreme type curvatures should be avoided". Following these criteria, Edmondson et al. [Edm+96] present a framework that particularly places straight labels along single line features. Further, Wolff et al. [Wol+00] also consider the case that labels may bend. Imhof [Imh75] mostly expands on single line features such as rivers leaving out the placement of labels in road maps.

For road maps it is common practice to draw road sections as curves of certain widths. Labels are embedded inside the shape of the curves following their curvature.

Based on interviews with cartographers, Chirié [Chi00] presents criteria for label placement in road maps. These criteria include that

(C1) labels are placed inside and parallel to the road shapes,
(C2) every road section between two junctions should be clearly identified, and
(C3) no two road labels may intersect.

Chirié [Chi00] and Strijk [Str01, Ch. 9] present simple, local heuristics that place non-overlapping labels based on a discrete set of candidate positions. Seibert and Unger [SU02] utilize the geometric properties of grid-based road networks and prove that it is NP-complete to decide whether at least one label can be placed for each road. For the same grid-based setting Neyer and Wagner [NW00] evaluate a practically efficient algorithm that is not applicable for general road networks.

Area Feature Labeling. Since we do not consider area features in this thesis, we only give a short overview on this feature type. Van Roessel [Roe89] presents a simple heuristic for computing a set of maximal rectangles completely contained in a given polygon. One of these rectangles is then used for the label placement. Pinto and Freeman [PF96] present a *feedback approach* that, based on an initial label placement, improves the result according to cartographic criteria. Edmondson et al. [Edm+96] incorporate a heuristic for placing labels in area features into a generic framework. Rylov and Reimer [RR16] describe an algorithm for placing labels outside the area features. Further, van Goethem et al. [GKS16] present algorithms for labeling island groups, which is closely related to area feature labeling. Summarizing, only little attention has been payed to automatic label placement for area features.

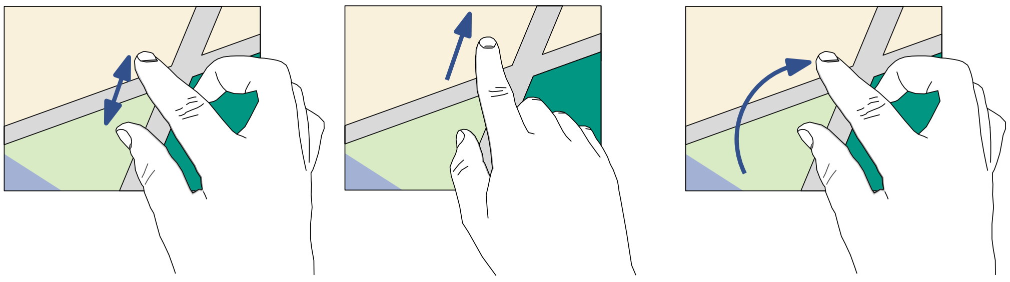
3.2 Dynamic Map Labeling[1]

In contrast to traditional static maps, dynamic digital maps support continuous movement of the map viewport based on panning, rotation, or zooming; see Figure 3.2 for a schematic illustration of interactive maps. Creating smooth visualizations under such map dynamics induces challenging geometric problems, e.g., continuous generalization [SB04] or *dynamic map labeling*.

In 2003, Petzold et al. [PGP03] presented a framework for automatically placing labels on dynamic maps. They split the label placement procedure into two phases, namely a (possibly time-consuming) pre-processing phase and a query phase which computes the labeling of custom-scale maps. However, this approach does not guarantee that labels do not *jump* or *flicker* while transforming the map.

In 2006, Been et al. [BDY06] introduced the first formal model for dynamic maps and dynamic labels, formulating a general optimization problem. They describe the

[1]This part is based on and partly taken from joint work with Lukas Barth, Martin Nöllenburg and Darren Strash[Bar+16].



(a) Zooming. **(b)** Panning. **(c)** Rotation.

Figure 3.2: Illustration of interactive maps providing *zooming*, *panning*, and *rotation* as user interaction.

change of a map by the operations *zooming*, *panning*, and *rotation*. In order to avoid *flickering* and *jumping* labels while transforming the map with zooming and panning, they require four desiderata for *consistent* dynamic map labeling. These comprise *monotonicity*, i.e., labels should not vanish when zooming in or appear when zooming out (or any of the two when panning), *invariant point placement*, i.e., label positions and size remain invariant during movement, and *history independence*, i.e., placement and selection of labels should be a function of the current map state only. Monotonicity is modeled as selecting for each label at most one scale interval, the so-called *active range*, during which the label is displayed. They introduce the *active range optimization problem* (ARO) maximizing the sum of active ranges over all labels such that no two labels overlap and all desiderata are fulfilled. They prove that ARO is NP-hard for star-shaped labels and present a greedy algorithm that computes an optimal solution for a simplified variant in polynomial time.

That model is the starting point for several subsequent papers considering the operations *zooming*, *panning* and *rotation*, mostly independently. Been et al. [Bee+10] take a closer look at different variants of ARO for zooming. They show NP-hardness and give approximation algorithms. In the same manner further variants are investigated by Liao et al. [LLP14]. Gemsa et al. [GNR11] present a fully polynomial-time approximation scheme (FPTAS) for a special case of ARO, where the given map is one-dimensional and only zooming is allowed. However, they combine the selection problem with a placement problem in a slider model. Zhang et al. [Zha+15] also consider the model of Been et al. [BDY06] for zooming, however, instead of maximizing the total sum of active ranges, they maximize the minimum active range among all labels. They discuss similar variants as Liao et al. [LLP14] and Been et al. [Bee+10], and also prove NP-hardness and give approximation algorithms.

Gemsa et al. [GNR16a, GNR16b] extend the ARO model to *rotation* operations. They first show that the ARO problem is NP-hard in the considered setting and introduce an efficient polynomial-time-approximation scheme (FPTAS) for unit-height rectangles [GNR16a]. In subsequent work, they experimentally evaluate heuristics, algorithms with approximation guarantees and optimal approaches based on integer linear programming [GNR16b]. A similar setting for rotating maps is considered by Yokosuka and Imai [YI13]. However, instead of ARO, they aim at finding the maximum font size for which all labels can always be displayed without overlapping. Similarly to the ARO model, Funke et al. [FKS16] as well as Bahrdt et al. [Bah+17] modeled the dynamic map labeling problem as a set of prioritized balls in R^d whose radii grow linearly over time. They are then interested in the computation of an elimination order of the balls to avoid overlaps.

Apart from the results based on the consistency model of Been et al. [BDY06], other approaches and models are considered, too. Maass et al. [MD06] describe a view management system for interactive three-dimensional maps of cities also considering label placement. Mote [Mot07] presents a fast label placement strategy without a pre-processing phase. Luboschik [LSC08] describes a fast particle-based strategy that locally optimizes the label placement. None of these approaches takes consistency criteria for dynamic map labeling into account.

A different generalization of static point labeling is dynamic point labeling. Instead of transforming the map, the point set changes over time by adding or removing points as well as by moving points continuously. Inspired by air-traffic control, De Berg and Gerrits [BG13] consider moving points on a static map that must be labeled. They present a sophisticated heuristic for finding a reasonable trade-off between label speed and label overlap. Finally, Buchin and Gerrits [BG14] show that dynamic point labeling is strongly PSPACE-complete.

Embedded labels for road maps are also considered for interactive and dynamic maps. Maass and Döllner [MD07] provide a heuristic for labeling interactive 3D road maps taking obstacles into account. Vaaraniemi et al. [VTW12] present a study on a force-based labeling algorithm for dynamic maps considering both point and line features. Schwartges et al. [SWH14] investigate embedded labels in interactive maps allowing panning, zooming and rotation of the map. They evaluate a simple heuristic for maximizing the number of placed labels. In [Sch+15] Schwartges et al. take another approach using *billboards* (labels with short leaders) for naming roads in interactive 3D maps to avoid label distortion.

4 Label Placement in Road Maps: Model and Theory

Abstract. A road map can be interpreted as a graph embedded in the plane, in which each vertex corresponds to a road junction and each edge to a particular road section. In this chapter, we consider the cartographic problem to place non-overlapping road labels along the edges so that as many road sections as possible are identified by their name, i.e., covered by a label. We show that this is NP-hard in general, but the problem can be solved in $O(n^3)$ time if the road map is an embedded tree with n vertices. In the subsequent chapter we then show how to make both the model and the algorithm practicable.

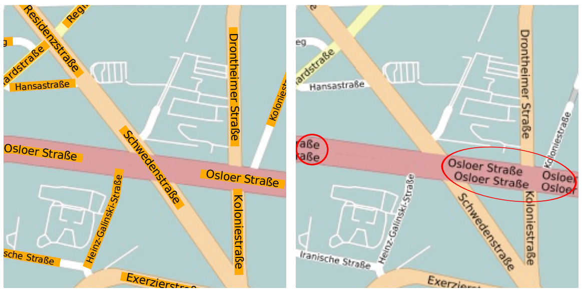
This chapter is based on and partly taken from joint work with Andreas Gemsa and Martin Nöllenburg [GNN14, GNN15, NN16].

4.1 Introduction

Road maps play an important role in daily life providing an easy-to-use tool for navigation. Their usefulness crucially relies on the placement of the roads' labels, which are typically contained in the interior of the depicted roads; see Figure 4.1 for an example. In order to create legible and appealing road maps automatically, Chirié [Chi00] elaborated general quality criteria for label placement based on interviews with cartographers; see also Chapter 3. For the reader's convenience we rephrase them below:

(C1) labels are placed inside and parallel to the road shapes,
(C2) every road section between two junctions should be clearly identified, and
(C3) no two road labels may intersect.

These criteria indicate that, in contrast to label placement for point features, maximizing the number of labels is not the appropriate objective for label placement of roads, since not every label that is placed necessarily contributes more information to the map. For example, consider the placed labels of the road *Osloer Straße* in Figure 4.1(b). We can easily remove some of those labels without losing any information, because a user can still identify the same road sections; see Figure 4.1(a). In online map services, however, one can often find such redundant labels; see Figure 4.2 for two examples. Some roads may have unnecessarily many labels, which may in turn cause others to remain completely unlabeled. Hence, the user cannot identify such roads on the map, a real disadvantage if headed for that road.



(a) Good **(b)** Poor

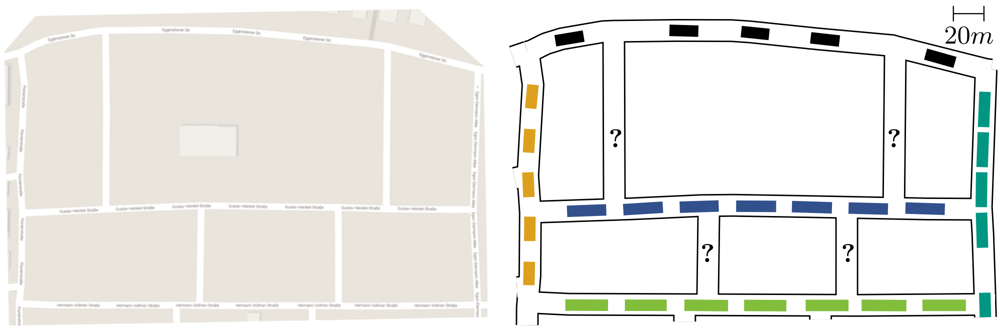
Figure 4.1: Road maps of Berlin. (a) Only few labels are placed to name the road sections. (b) A labeling produced by the OSM rendering engine Mapnik. It contains redundant labels naming the same road section multiple times. The six labels of road *Osloer Straße* are enclosed by red ellipses.

On the other hand, in contrast to grid-shaped road networks [SU02], placing a single label per road does not clearly identify all its road sections in general road networks either. Consider the example in Figure 4.3. In Figure 4.3(a), it is not obvious whether the depicted bridge in the center belongs to *Knuth St.* or to *Turing St.*

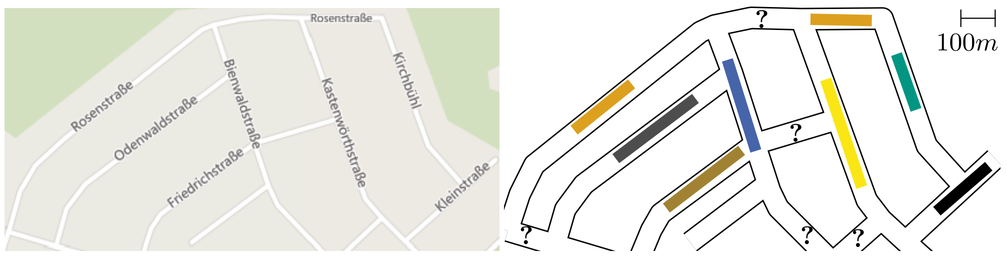
Contribution and Outline. Due to these observations, we do not aim at maximizing the number of labels, or the number of labeled roads, but the number of labeled *road sections*. For the purpose of this chapter, a *road section* forms a connected piece of the road network that logically belongs together, e.g., a part of a road between two junctions or a part that stands out by its color or width. Our model, however, is independent of the actual definition of road sections; any partition of the road network into disjoint road sections can be handled.

In contrast to the approaches of Chirié [Chi00] and Strijk et al. [Str01, Ch. 9], we consider label placement in road maps globally, applying a continuous sliding model. More precisely, as the underlying model, we introduce a new and versatile planar graph model based on criteria (C1)–(C3); see Section 4.2. Geometrically, a *road map* is the representation of a *road graph* G as an arrangement of *fat* curves in the plane $\mathbb{R}^2$. Each *road* is a connected subgraph of G (typically a simple path) and each edge belongs to exactly one road. Roads may intersect each other in *junctions*, the vertices of G, and we denote an edge in G as a *road section*. In road labeling, the task is to place the road names inside the fat curves so that the road sections are identified unambiguously, see Figure 4.3. We say that a road section is *labeled* if a label (partly) covers it.

In this chapter, we take an algorithmic, mathematical perspective on the optimization problem of maximizing the number of labeled road sections. In Section 4.3 we show that the problem of maximizing the number of labeled road sections is NP-hard for general road graphs, even if every road is a path. For the special case that the road graph is a tree, we present a polynomial-time algorithm in Section 4.4. This special case is not only of theoretical interest, but our algorithm in fact provides a very useful subroutine in exact or heuristic algorithms for labeling general road graphs, as we demonstrate in Chapter 5.



(a) Google Maps: Hermann-Vollmerstraße, Karlsruhe (Germany)



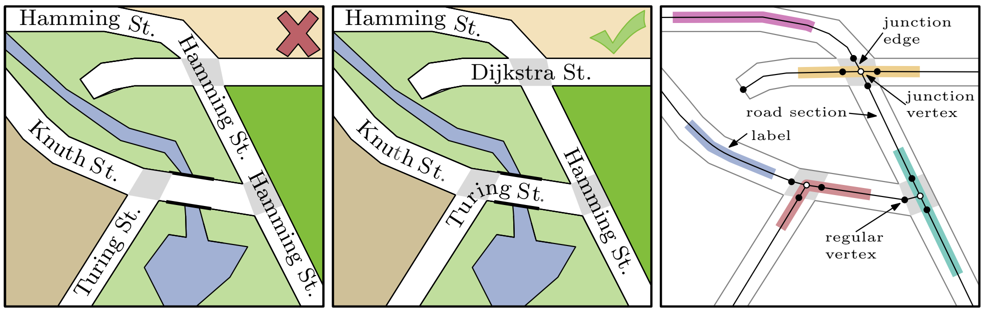
(b) Bing Maps: Kirchbühl, Karlsruhe (Germany)

Figure 4.2: Two maps of the map services Google Maps and Bing Maps. Left: Screenshot of a road map. Right: The redrawn road map to emphasize the label positions. Labels of the same color belong to the same road. (a) Labels are placed tightly packed in a row. While some roads have more labels then necessary, other roads are not labeled. (b) Road map consists of several unlabeled road sections.

4.2 Model

As argued, a road map is a collection of fat curves in the plane, each representing a particular piece of a named road. If two (or more) such curves intersect, they form junctions. A *road label* is again a fat curve (the bounding shape of the road name) that is contained in and parallel to the fat curve representing its road. We observe that labels of different roads can intersect only within junctions and that the actual width of the curves is irrelevant, except for defining the shape and size of the junctions. We use these observations to define the following abstract road graph model. In Chapter 5 we explain how to extract an abstract road graph from a given road network such that it can be used in practice.

A *road map* $\mathcal{M}$ is a planar *road graph* $G = (V, E)$ together with a planar embedding $\mathcal{E}(G)$, which can be thought of as the geometric representation of the road axes as thin curves; see Figure 4.3(c). We denote the number of vertices of G by n, and the number of edges by m. Observe that since G is planar $m = O(n)$. Each edge $e \in E$ is



(a) **(b)** **(c)**

Figure 4.3: (a)–(b): Two ways to label the same road network. Junctions are marked gray. Figure (b) labels all road sections. (c) Illustration of the road graph and relevant terms.

either a *road section*, which is not part of a junction, or a *junction edge*, which is part of a junction. Each vertex $v \in V$ is either a *junction vertex* incident only to junction edges, or a *regular vertex* incident to one road section and at most one junction edge, which implies that each regular vertex has degree at most two. A junction vertex v and its incident edges are denoted as a *junction*. The edge set E decomposes into a set $\mathcal{R}$ of edge-disjoint *roads*, where each road $R \in \mathcal{R}$ induces a connected subgraph of G. Without loss of generality we assume no two road sections in G are incident to the same vertex. Thus, a road decomposes into road sections, separated by junction vertices and their incident junction edges. In realistic road networks the number of roads connected passing through a junction is small and does not depend on the size of the road network. We therefore assume that each vertex in G has constant degree. We assume that each road $R \in \mathcal{R}$ has a name whose length we denote by $\lambda(R)$.

For simplicity, we identify the embedding $\mathcal{E}(G)$ with the points in the plane covered by $\mathcal{E}(G)$, i.e. $\mathcal{E}(G) \subseteq \mathbb{R}^2$. We also use $\mathcal{E}(v)$, $\mathcal{E}(e)$, and $\mathcal{E}(R)$ to denote the embeddings of a vertex v, an edge e, and a road R.

We model a label as a simple open curve $\ell\colon [0,1] \to \mathcal{E}(G)$ in $\mathcal{E}(G)$. Unless mentioned otherwise, we consider a curve ℓ always to be simple and open, i.e., ℓ has no self-intersections and its end points do not coincide. In order to ease the description, we identify a curve ℓ in $\mathcal{E}(G)$ with its image, i.e., ℓ denotes the set $\{\ell(t) \in \mathcal{E}(G) \mid t \in [0,1]\}$. The start point of ℓ is denoted as the *head* $h(\ell)$ and the endpoint as the *tail* $t(\ell)$. The length of ℓ is denoted by $\operatorname{length}(\ell)$. The curve ℓ (partly) *covers* a road section r if $\ell \cap \mathcal{E}(r) \neq \emptyset$. In that case we say that ℓ *labels* the road section r. For a set $\mathcal{L}$ of curves $\omega(\mathcal{L})$ is the number of road sections that are labeled by the curves in $\mathcal{L}$. For a single curve ℓ we use $\omega(\ell)$ instead of $\omega(\{\ell\})$. For two curves ℓ_1 and ℓ_2 it is not necessarily true that $\omega(\{\ell_1, \ell_2\}) = \omega(\ell_1) + \omega(\ell_2)$, because they may label the same road section twice.

A *label* ℓ for a road R is a curve $\ell \subseteq \mathcal{E}(R)$ of length $\lambda(R)$ whose endpoints must lie on road sections and not on junction edges or junction vertices. Requiring that labels end on road sections avoids ambiguous placement of labels in junctions where it is unclear how the road passes through it. A *labeling* $\mathcal{L}$ for a road map with road set $\mathcal{R}$ is a set of mutually non-overlapping labels, where we say that two labels ℓ and ℓ' *overlap* if they intersect in a point that is not their respective head or tail.

Following the cartographic quality criteria (C1)–(C3), our goal is to find a labeling $\mathcal{L}$ that maximizes the number of labeled road sections, i.e., for any labeling $\mathcal{L}'$ we have $\omega(\mathcal{L}') \leq \omega(\mathcal{L})$. We call this problem MaxLabeledRoads.

Note that assuming the road graph G to be planar is not a restriction in practice. Consider for example a road section r that overpasses another road section r', i.e., r is a bridge over r', or r' is a tunnel underneath r. In order to avoid overlaps between labels placed on r and r', we either can model the intersection of r and r' as a regular crossing of two roads or we split r' in smaller road sections that do not cross r. In both cases the corresponding road graph becomes planar. In the latter case we may obtain more independent roads created by chopping r' into smaller pieces.

4.3 Computational Complexity

We first study the computational complexity of road labeling and prove NP-hardness of MaxLabeledRoads in the following sense.

Theorem 4.1. *For a given road map $\mathcal{M}$ and an integer K it is NP-hard to decide if in total at least K road sections can be labeled.*

Proof. We perform a reduction from the NP-complete planar monotone 3-Sat problem [Lic82]. An instance of planar monotone 3-Sat is a Boolean formula φ with n variables and m clauses (disjunctions of at most three literals) that satisfies the following additional requirements: (i) φ is *monotone*, i.e., every clause contains either only positive literals or only negative literals and (ii) the induced variable-clause graph H_φ of φ is planar and can be embedded in the plane with all variable vertices on a horizontal line, all positive clause vertices on one side of the line, all negative clauses on the other side of the line, and the edges drawn as rectilinear curves connecting clauses and contained variables on their respective side of the line. We construct a road map $\mathcal{M}_\varphi$ that mimics the shape of the above embedding of H_φ by defining variable and clause gadgets, which simulate the assignment of truth values to variables and the evaluation of the clauses. We refer to Figure 4.4 for a sketch of the construction.

Chain Gadget. The basic building block is the *chain gadget*, which consists of an alternating sequence of equally long horizontal and vertical roads with identical label lengths that intersect their respective neighbors in the sequence and form junctions with them as indicated in Figure 4.4(c). Assume that the chain consists of $k \geq 3$ roads. Then each road except the first and last one decomposes into three road sections split

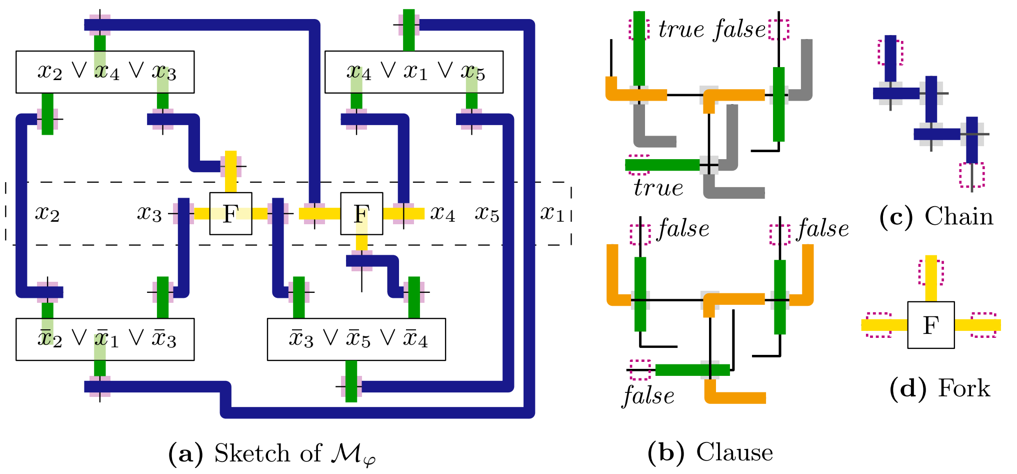


Figure 4.4: Illustration of NP-hardness proof. (a) 3-Sat formula $\varphi = (x_4 \vee x_1 \vee x_5) \wedge (x_2 \vee x_4 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee \bar{x}_4)$ represented as road graph $\mathcal{M}_\varphi$. Truth assignment is $x_1 =$ *true*, $x_2 =$ *true*, $x_3 =$ *false*, $x_4 =$ *false* and $x_5 =$ *false*. (b) Clause gadget in two states. (c) The chain is the basic building block for the proof. (d) Schematized fork gadget.

by two junctions, a longer central section and two short end sections; the first and last road consist of only two road sections, a short one and a long one, separated by one junction. (These two roads will later be connected to other gadgets; indicated by dotted squares in Figure 4.4(c).) The label length and distance between junctions is chosen so that for each road either the central and one end section is labeled, or no section at all is labeled. For the first and last road, both sections are labeled if the junction is covered and otherwise only the long section can be labeled. We have k roads and $k-1$ junctions. Each label must block a junction, if it labels two sections. So the best possible configuration blocks all junctions and labels $2(k-1)+1 = 2k-1$ road sections.

The chain gadget has exactly two states, in which $2k-1$ road sections are labeled. Either the label of the first road does not block a junction and labels a single section and all subsequent roads have their label cover the junction with the preceding road in the sequence, or the label of the last road does not block a junction and all other roads have their label cover the junction with the successive road in the sequence. In any other configuration there is at least one road without any labeled section and thus at most $2k-2$ sections are labeled. We use the two optimal states of the gadget to represent and transmit the values *true* and *false* from one end to the other.

Fork Gadget. The *fork gadget* allows us to split the value represented in one chain into two chains, which is needed to transmit the truth value of a variable into multiple

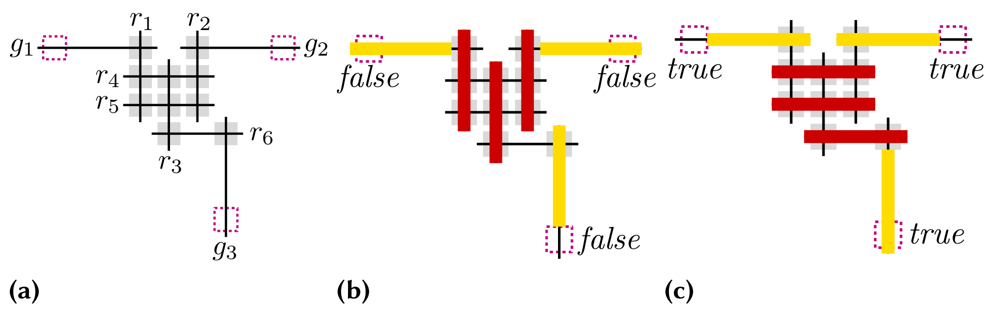


Figure 4.5: Illustration of the fork gadget. (a) Structure of the fork gadget. (c) Configuration transmitting the value *false*. (b) Configuration transmitting the value *true*.

clauses. To that end it connects to an end road of three chain gadgets by sharing junctions.

The core of the fork consists of six roads $r_1, \ldots, r_6$, whereas r_1, r_2, and r_3 are vertical line segments and r_4, r_5 and r_6 are horizontal line segments; see Figure 4.5. We arrange those roads such that r_1 and r_2 have each one junction with r_4 and one junction with r_5. Further, r_3 has one junction with r_4, one with r_5 and one with r_6. The label length of those roads is chosen so that it is exactly the length of the roads. Hence, a placed label labels all road sections of the roads.

Further, there are three roads g_1, g_2, g_3 such that g_1 has one junction with r_1, g_2 has one junction with r_2 and g_3 has one junction with r_6. In all three cases we place the junction so that it splits the road in a short road section that is shorter than the road's label length and a long road section that has exactly the road's label length. We call g_1, g_2 and g_3 *gates*, because later these roads will be connected to the end roads of chains by junctions. To that end those *connecting* junctions will be placed on the long road sections of the gates; see violet dotted areas in Figure 4.5.

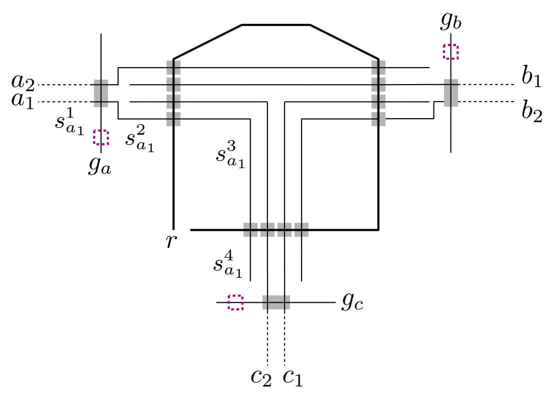
The fork gadget has exactly two states, in which 16 road sections are labeled. In the first state the labels of r_1, r_2 and r_3 are placed; see Figure 4.5(b). Hence, the labels of g_1 and g_2 label only the long road sections of g_1 and g_2, but not the short ones. The label of g_3 labels both the long and short road section of g_3. In the second state the labels of r_4, r_5, r_6 are placed; see Figure 4.5(c). Hence, the labels of g_1 and g_2 label the long and short road sections of g_1 and g_2, while only the long road section of g_3 is labeled. In any other configuration fewer road sections are labeled. We use the two optimal states of the gadget to represent and transmit the values *true* and *false* from one gate to the other two gates. More specifically the gates g_1 and g_2 are connected with chains that lead to the same literal, while g_3 is connected with a chain that leads to the complementary literal.

Variable Gadget. We define the *variable gadgets* simply by connecting chain and fork gadgets into a connected component of intersecting roads. This construction already has the functionality of a variable gadget: it represents (in a labeling identifying the maximum number of road sections) the same truth value in all of its branches, synchronized by the fork gadgets, see the blue chains and yellow forks in Figure 4.4(a). More precisely, we place a sequence of chains linked by fork gadgets along the horizontal line on which the variable vertices are placed in the drawing H_φ. Each fork creates a branch of the variable gadget either above or below the line. We create as many branches above (below) the line as the variable has occurrences in positive (negative) clauses in φ. The first and last chain on the line also serve as branches. The synchronization of the different branches via the forks is such that either all top branches have their road labels pushed away from the line and all bottom branches pulled towards the line or vice versa. In the first case, we say that the variable is in the state *false* and in the latter case that it is in the state *true*. The example in Figure 4.4 has two variables set to *true* and three variables set to *false*.

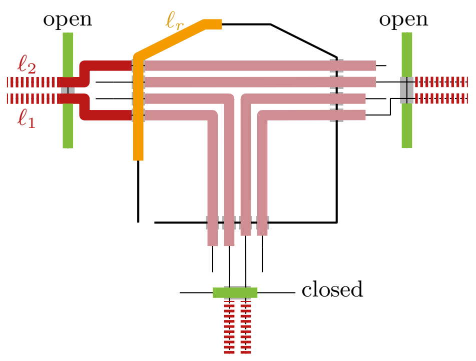
Clause Gadget. Finally, we need to create the clause gadget, which links three branches of different variables. The core of the gadget is a single road that consists of three subpaths meeting in one junction. Each subpath of that road shares another junction with one of the three incoming variable branches. Beyond each of these three junctions the final road sections are just long enough so that a label can be placed on the section. However, the section between the central junction of the clause road and the junctions with the literal roads is shorter than the label length. The road of the clause gadget has six sections in total and we argue that the six sections can only be labeled if at least one incoming literal evaluates to *true*. Otherwise at most five sections can be labeled. By construction, each road in the chain of a false literal has its label pushed towards the clause, i.e., it blocks the junction with the clause road. As long as at least one of these three junctions is not blocked, all sections can be labeled; see Figure 4.4(b). But if all three junctions are blocked, then only two of the three inner sections of the clause road can be labeled and the third one remains unlabeled.

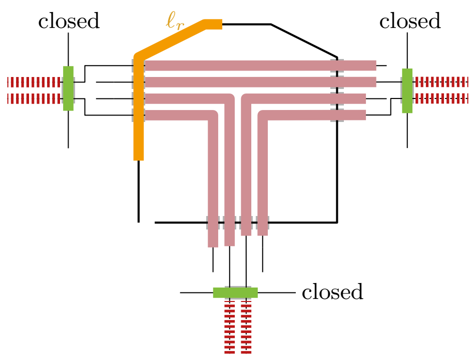
Reduction. Obviously, the size of the instance $\mathcal{M}_\varphi$ is polynomial in n and m. If we have a satisfying variable assignment for φ, we can construct the corresponding road labeling and the number of labeled road sections is six per clause and a fixed constant number K' of sections in the variable gadgets, i.e., at least $K = K' + 6m$. On the other hand, if we have a road labeling with at least K labeled sections, each variable gadget is in one of its two maximum configurations and each clause road has at least one label that covers a junction with a literal road, meaning that the corresponding truth value assignment of the variables is indeed a satisfying one. This concludes the reduction. $\square$

Since MaxLabeledRoads is an optimization problem, we only present the NP-hardness proof. Still, one can argue that the corresponding decision problem is NP-complete by guessing which junctions are covered by which label and then using linear programming for computing the label positions. We omit the technical details. Further,



(a) Structure.





(b) One literal is *true*. **(c)** All literals are *false*.

Figure 4.6: Illustration of alternative clause gadget, which only uses paths as roads. (a) Structure of the clause gadget. (b) Optimal labeling for the case that at least one literal is *true*. (c) Optimal labeling for the case that all literals are *false*.

most roads in the reduction are paths, except for the central road in each clause gadget, which is a degree-3 star. In fact, we can strengthen Theorem 4.1 by using a more complex clause gadget instead that uses only paths as described as follows.

Alternative Clause Gadget. In this section we describe a clause gadget that can be used as an alternative to the one presented in the previous section. Since it consists only of roads that are paths, this gadget strengthens Theorem 4.1.

Theorem 4.2. *For a given road map $\mathcal{M}$ and an integer K it is NP-hard to decide if in total at least K road sections can be labeled, even if all roads are paths.*

The clause gadget consists of ten roads, r, g_a, g_b, g_c ,a_i, b_i and c_i with $i \in \{1, 2\}$ that all are paths; see Figure 4.6. Going along r from one end to the other, the junctions with the roads a_i, b_i and c_i ($1 \leq i \leq 2$) occur in three densely packed blocks. The blocks are described by the sequence of roads intersecting r. The first block is $B_a = (a_1, c_2, b_1, a_2)$, the second block is $B_b = (a_2, b_1, c_1, b_2)$ and the third block is $B_c = (b_2, c_1, c_2, a_1)$. The label length of r is chosen so that at most three labels can be placed on r, but each road section is shorter than a label of r. Choosing the length of the road sections appropriately, we further ensure that we can place a label that crosses all junctions of one of the blocks without crossing the junctions of another block.

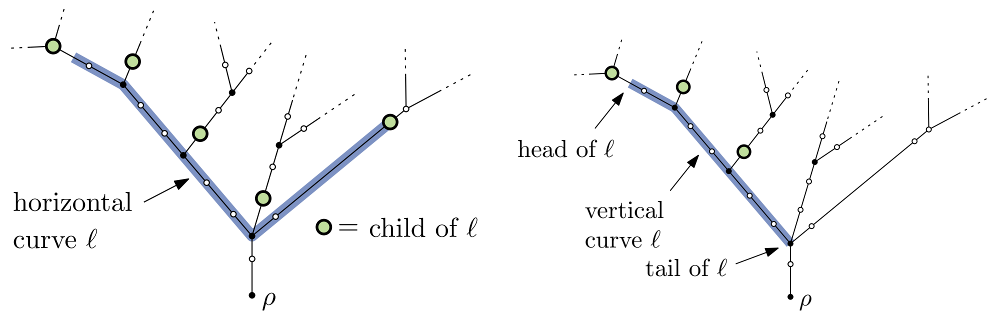
We now describe junctions of the roads g_a, g_b, g_c ,a_i, b_i and c_i with $i \in \{1, 2\}$. The road a_1 first intersects g_a and then r twice. Let $s^1_{a_1}$, $s^2_{a_1}$, $s^3_{a_1}$ and $s^4_{a_1}$ denote these road sections in that particular order. The length of $s^1_{a_1}$ is chosen so that a single label can be placed on $s^1_{a_1}$, while the others are shorter than the label length of a_1. More specifically, we define a_1's label length such that a label covers the sections in either $\{s^1_{a_1}\}$, $\{s^1_{a_1}, s^2_{a_1}\}$, $\{s^1_{a_1}, s^2_{a_1}, s^3_{a_1}\}$, $\{s^2_{a_1}, s^3_{a_1}, s^4_{a_1}\}$ or $\{s^3_{a_1}, s^4_{a_1}\}$. We define the intersections and the label length for a_2, analogously. Further, g_a intersects a_1 and a_2 in one junction, i.e., the edge of g_a connecting both junction vertices is a junction edge. The label length of g_a is chosen so that a label can cross g_a's only junction. The length of g_a's road sections is at least as long as g_a's label length. We call g_a a *gate*, because later this road will be connected to the end road of a chain by a junction; see violet square in Figure 4.6(a). For b_1, b_2, c_1, c_2 we introduce analogous junctions and road sections, however, b_1 and b_2 intersect g_b instead of g_a, and c_1 and c_2 intersect g_c instead of g_a.

In order to label both road sections of a gate, either two labels can be placed on the road sections separately, or one label that goes through the junction. In the former case the gate is *open* and in latter case it is *closed*; see Figure 4.6(b). We observe that it only makes sense to close a gate, if at least one road section of the gate does not allow to place a label that is only contained in that road section. This case will occur if and only if the connected chain transmits the value *false* to the clause.

Assume that at least one gate is open, i.e., one literal of the clause is true; see Figure 4.6(b). Without loss of generality let g_a be open. We place a label ℓ_r on r such that it crosses the junctions of block B_a and labels 5 sections. Since g_a is open, we can place a label ℓ_1 that labels $s^1_{a_1}$ and $s^2_{a_1}$. Analogously, we can place a label ℓ_2 labeling $s^1_{a_2}$ and $s^2_{a_2}$. Placing further labels as indicated in Figure 4.6(b), we label five road sections of r and all road sections of any other road except for $s^4_{c_2}$, $s^4_{b_1}$. Hence, 33 road sections are labeled.

We observe that we can place the labels of b_1, b_2, c_1, c_2 such that they do not cross the junctions of g_b and g_c, respectively. Hence, it does not matter whether g_b and g_c are closed or open, i.e., it does not matter whether the corresponding literals are *true* or *false*.

We now argue that this is an optimal labeling. If $s^4_{c_2}$ or $s^4_{b_1}$ were labeled, the label ℓ_r must be placed such that the junctions of r with c_2 and b_1 are not crossed, respectively. This decreases the number of labeled road sections as least as much labeling $s^4_{c_2}$ and



(a) Horizontal label. **(b)** Vertical label.

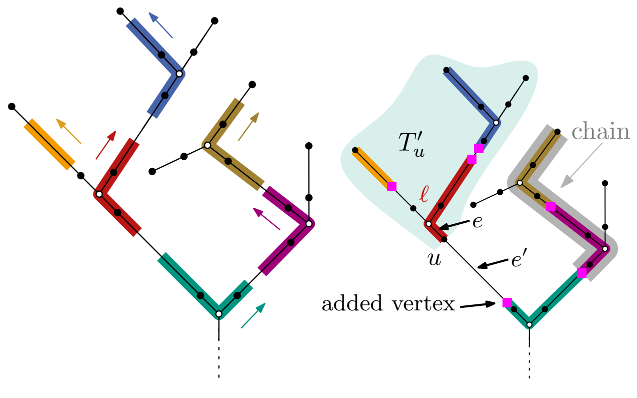
Figure 4.7: Basic definitions of horizontal and vertical labels.

$s^4_{b_1}$ increases the number of labeled road sections. In order to label at least one of the unlabeled road sections of r, we need to place a label that crosses B_b or B_c. Obviously, this yields a smaller number of labeled road sections than 31.

Finally, assume that all gates are closed; see Figure 4.6(c). Consider, the same labeling as before. However, this time we cannot label $s^2_{a_1}$ and $s^2_{a_2}$ anymore. Hence, this labeling has only 29 labeled road sections. Obviously, it cannot be improved by changing the placement of the remaining labels or adding labels.

4.4 An Efficient Algorithm for Tree-Shaped Road Maps

In this section we assume that the underlying road graph of the road map is a tree $T = (V, E)$. In Section 4.4.1 we present a polynomial-time algorithm to optimally solve MaxLabeledRoads for trees; Section 4.4.2 shows how to improve its running time and space consumption. Our approach uses the basic idea that removing the vertices, whose embeddings lie in a curve $c \subseteq \mathcal{E}(T)$, splits the tree into independent parts. In particular this is true for labels. We assume that T is rooted at an arbitrary leaf ρ and that its edges are directed away from ρ; see Figure 4.7. For two points $p, q \in \mathcal{E}(T)$ we define $\mathrm{d}(p, q)$ as the length of the shortest curve in $\mathcal{E}(T)$ that connects p and q. For two vertices u and v of T we also write $\mathrm{d}(u, v)$ instead of $\mathrm{d}(\mathcal{E}(u), \mathcal{E}(v))$. For a point $p \in E(T)$ we abbreviate the distance $\mathrm{d}(p, \rho)$ to the root ρ by d_p. For a curve ℓ in $\mathcal{E}(T)$, we call $p \in \ell$ the *lowest point* of ℓ if $\mathrm{d}_p \leq \mathrm{d}_q$, for any $q \in \ell$. As T is a tree, p is unique. We distinguish two types of curves in $\mathcal{E}(T)$. A curve ℓ is *vertical* if $h(\ell)$ or $t(\ell)$ is the lowest point of ℓ; otherwise we call ℓ *horizontal*; see Figure 4.7. Without loss of generality we assume that the lowest point of each vertical curve ℓ is its tail $t(\ell)$. Since labels are modeled as curves, they are also either vertical or horizontal. For a vertex $u \in V$ let T_u denote the subtree rooted at u.



(a) Pushing labels. **(b)** Canonical labeling.

Figure 4.8: Illustration of canonical labelings. (a) Each label is moved away from the root as far as possible while its head and tail must remain on their respective road sections. (b) The canonical labeling obtained of (a). The tree is subdivided by additional vertices (pink squares) at the tails and heads of the labels.

4.4.1 Basic Approach

We first determine a finite set of candidate positions for the heads and tails of labels, and transform T into a tree $T' = (V', E')$ by subdividing some of T's edges so that it contains a vertex for every candidate position. To that end we construct for each regular vertex $v \in V$ a chain of tightly packed vertical labels that starts at $\mathcal{E}(v)$, is directed towards ρ, and ends when either the road ends, or adding the next label does not increase the number of labeled road sections. More specifically, we place a first vertical label ℓ_1 such that $h(\ell_1) = \mathcal{E}(v)$. For $i = 2, 3, \ldots$ we add a new vertical label ℓ_i with $h(\ell_i) = t(\ell_{i-1})$, as long as $h(\ell_i)$ and $t(\ell_i)$ do not lie on the same road section and none of ℓ_i's endpoints lie on a junction edge. We use the tails of all those labels to subdivide the tree T. Doing this for all regular vertices of T we obtain the tree T', which we call the *subdivision tree* of T. The vertices in $V' \setminus V$ are neither junction vertices nor regular vertices. Since each chain consists of $O(n)$ labels the cardinality of V' is $O(n^2)$. We call an optimal labeling $\mathcal{L}$ of T a *canonical labeling* if for each label $\ell \in \mathcal{L}'$ there exists a vertex v in T' with $\mathcal{E}(v) = h(\ell)$ or $\mathcal{E}(v) = t(\ell)$. The next lemma proves that it is sufficient to consider canonical labelings.

Lemma 4.1. *For any road graph T that is a tree, there exists a canonical labeling $\mathcal{L}$.*

Proof. Let $\mathcal{L}$ be an optimal labeling of T. We *push* the labels of $\mathcal{L}$ as far as possible towards the leaves of T without changing the labeled road sections; see Figure 4.8. More specifically, starting with the labels closest to the leaves, we move each label away from the root as far as possible while its head and tail must remain on their respective road sections. For a vertical label this direction is unique, while for horizontal labels we can choose any of the two. Then, for each label its head or tail either coincides with a leaf of T, with some internal regular vertex, or with the head of another label. Consequently, each vertical label belongs to a chain of tightly packed vertical labels starting at a regular vertex $v \in V$. Further, the head or tail of each horizontal label coincides with the end of a chain of tightly packed vertical labels or a regular vertex of T, which proves the claim. □

We now explain how to construct such a canonical labeling. To that end we first introduce some notations. For a vertex $u \in V'$ let $\mathcal{L}(u)$ denote a labeling that labels a maximum number of road sections in T only using valid labels in $\mathcal{E}(T'_u)$, where T'_u denotes the subtree of T' rooted at u. Note that those labels also may cover the incoming road section of u, e.g., label ℓ in Figure 4.8(b) covers the edge e'.

Further, the children of a vertex $u \in V'$ are denoted by the set $N(u)$; we explicitly exclude the parent of u from $N(u)$. Further, consider an arbitrary curve ℓ in $\mathcal{E}(T)$ and let $\ell' = \ell \setminus \{t(\ell), h(\ell)\}$. We observe that removing all vertices of T' contained in ℓ' together with their incident outgoing edges creates several independent subtrees. We call the roots of these subtrees (except the one containing ρ) *children* of ℓ (see Figure 4.7). If no vertex of T' lies in ℓ', the curve is contained in a single edge $(u, v) \in E'$. In that case v is the only child of ℓ. We denote the set of all children of ℓ as $N(\ell)$.

For each vertex u in T' we introduce a set $C(u)$ of *candidates*, which model potential labels with lowest point $\mathcal{E}(u)$. If u is a regular vertex of T or $u \in V' \setminus V$, the set $C(u)$ contains all vertical labels ℓ with lowest point $\mathcal{E}(u)$. If u is a junction vertex, $C(u)$ contains all horizontal labels that start or end at a vertex of T' and whose lowest point is $\mathcal{E}(u)$. In both cases we assume that $C(u)$ also contains the degenerated curve $\bot_u = \mathcal{E}(u)$, which is the *dummy label* of u. We set $N(\bot_u) = N(u)$ and $\omega(\bot_u) = 0$.

For a curve ℓ we define $\mathcal{L}(\ell) = \bigcup_{v \in N(\ell)} \mathcal{L}(v) \cup \{\ell\}$. Thus, $\mathcal{L}(\ell)$ is a labeling comprising ℓ and the labels of its children's optimal labelings. We call a label $\overline{\ell} \in C(u)$ with $\overline{\ell} = \operatorname{argmax}\{\omega(\mathcal{L}(\ell)) \mid \ell \in C(u)\}$ an *optimal candidate* of u. Next, we prove that it is sufficient to consider optimal candidates to construct a canonical labeling.

Lemma 4.2. *Given a vertex u of T' and an optimal labeling $\mathcal{L}(u)$ and let $\overline{\ell}$ be an optimal candidate of u, then it is true that $\omega(\mathcal{L}(u)) = \omega(\mathcal{L}(\overline{\ell}))$.*

Proof. First note that $\omega(\mathcal{L}(u)) \geq \omega(\mathcal{L}(\overline{\ell}))$ because both labelings $\mathcal{L}(u)$ and $\mathcal{L}(\overline{\ell})$ only contain labels that are embedded in $\mathcal{E}(T'_u)$. By Lemma 4.1 we can assume without loss of generality that $\mathcal{L}(u)$ is a canonical labeling. Let ℓ be the label of $\mathcal{L}(u)$ with $\mathcal{E}(u)$ as the lowest point of ℓ (if it exists).

If ℓ exists, then the vertices in $N(\ell)$ are roots of independent subtrees, which directly yields $\omega(\mathcal{L}(u)) = \omega(\mathcal{L}(\ell))$. By construction of $C(u)$ we further know that ℓ is contained in $C(u)$. Hence, ℓ is an optimal candidate of u, which implies $\omega(\ell) = \omega(\overline{\ell})$.

If ℓ does not exist, then we have

$$\omega(\mathcal{L}(u)) = \omega(\bigcup_{v \in N(u)} \mathcal{L}(v)) \overset{(1)}{=} \omega(\bigcup_{v \in N(\bot_u)} \mathcal{L}(v) \cup \{\bot_u\}) = \omega(\mathcal{L}(\bot_u)).$$

Equality (1) follows from $N(\bot_u) = N(u)$ and the definition that $\bot_u$ does not label

Algorithm 1: Computing an optimal labeling of T.

```
  Input: Road graph $T$, where $T$ is a tree with root $\rho$.
  Output: Optimal labeling $\mathcal{L}(\rho)$ of $T$.
1 $T' \leftarrow$ compute subdivision tree of $T$
2 for each leaf $v$ of $T'$ do $\mathcal{L}(v) \leftarrow \emptyset$
3 for each vertex $u$ of $T'$ considered in a bottom-up traversal of $T'$ do
4     $\mathcal{L}(u) \leftarrow \mathcal{L}(\texttt{OptCandidate}(u))$
5 return $\mathcal{L}(\rho)$
```

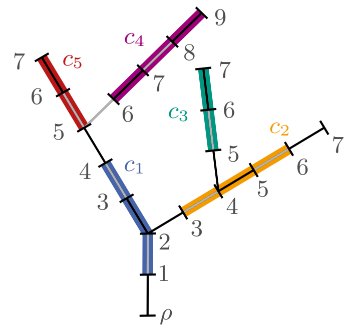
any road section. Since $\perp_u$ is contained in $C(u)$, the dummy label $\perp_u$ is the optimal candidate $\overline{\ell}$. □

Algorithm 1 first constructs the subdivision tree $T' = (V', E')$ from T. Then starting with the leaves of T' and going to the root ρ of T', it computes an optimal candidate $\overline{\ell}$ =`OptCandidate`(u) for each vertex $u \in V'$ in a bottom-up fashion. By Lemma 4.2 the labeling $\mathcal{L}(\overline{\ell})$ is an optimal labeling of T'_u. In particular $\mathcal{L}(\rho)$ is the optimal labeling of T.

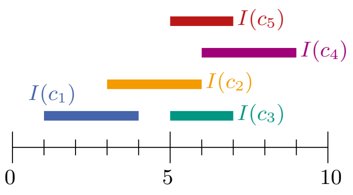
Due to the size of the subdivision tree T' we consider $O(n^2)$ vertices. Implementing `OptCandidate`(u), which computes an optimal candidate $\overline{\ell}$ for u, naively, creates $C(u)$ explicitly. We observe that if u is a junction vertex, $C(u)$ may contain $O(n^2)$ labels; $O(n^2)$ pairs of road sections of different subtrees of u can be connected by horizontal labels. Each label can be constructed in $O(n)$ time using a breadth-first search. Thus, for each vertex u the procedure `OptCandidate` needs in a naive implementation $O(n^3)$ time, which yields $O(n^5)$ running time in total. Further, we need $O(n^2)$ storage to store T'. Note that we do not need to store $\mathcal{L}(u)$ for each vertex u of T', but by Lemma 4.2 we can reconstruct it using $\mathcal{L}(\overline{\ell})$, where $\overline{\ell}$ is the optimal candidate of u. To that end we store for each vertex of T' its optimal candidate $\overline{\ell}$ and $w(\mathcal{L}(\overline{\ell}))$.

Theorem 4.3. *For a road map with a tree as underlying road graph,* MaxLabeledRoads *can be solved in $O(n^5)$ time using $O(n^2)$ space.*

In case that all roads are paths, Algorithm 1 runs in $O(n^4)$ time, because for each $u \in V'$ the set $C(u)$ contains $O(n)$ labels. Further, besides the *primary objective* to label a maximum number of road sections, Chirié [Chi00] also suggested several additional *secondary objectives*, e.g., labels should overlap as few junctions as possible. Our approach allows us to easily incorporate secondary objectives by changing the weight function ω appropriately.



(a) Tree representation.



(b) Interval representation.

Figure 4.9: Superposing curves. (a) E.g., c_1 and c_2 superpose each other, while c_1 and c_5 do not. The tree is annotated with distance marks. (b) Two curves superpose each other if their distance intervals intersect.

4.4.2 Improvements on Running Time

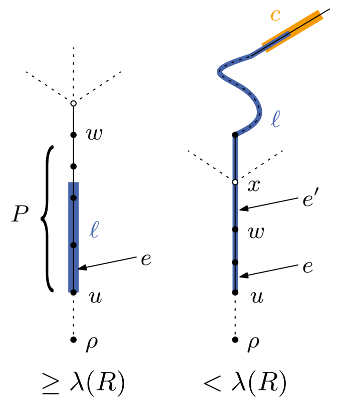
In this part we describe how the running time of Algorithm 1 can be improved to $O(n^3)$ time by speeding up `OptCandidate`(u) to $O(n)$ time.

For an edge $e = (u, v) \in E \cup E'$ we call a vertical curve $\ell \subseteq \mathcal{E}(T)$ an *e-rooted* curve, if $t(\ell) = \mathcal{E}(u)$, $h(\ell)$ is located on a road section, and $\text{length}(\mathcal{E}(e) \cap \ell) = \min\{\text{length}(\ell), \text{length}(\mathcal{E}(e))\}$, i.e., ℓ emanates from $\mathcal{E}(u)$ passing through e; for example the red label in Figure 4.8(b) is an e-rooted curve. An e-rooted curve ℓ is *maximal* if there is no other e-rooted curve ℓ' with $\text{length}(\ell) = \text{length}(\ell')$ and $\omega(\mathcal{L}(\ell')) > \omega(\mathcal{L}(\ell))$. We observe that in any canonical labeling each vertical label ℓ is a (u, v)-rooted curve with $(u, v) \in E'$, and each horizontal label ℓ can be composed of a (u, v_1)-rooted curve ℓ_1 and a (u, v_2)-rooted curve ℓ_2 with $(u, v_1), (u, v_2) \in E'$ and $\mathcal{E}(u)$ is the lowest point of ℓ; see Figure 4.10(a) and Figure 4.10(b), respectively. Further, for a vertical curve c in $\mathcal{E}(T)$ its *distance interval* $I(c)$ is $[\mathrm{d}_{t(c)}, \mathrm{d}_{h(c)}]$. Since T is a tree, for every point p of c we have $\mathrm{d}_p \in I(c)$. Two vertical curves c and c' *superpose* each other if $I(c) \cap I(c') \neq \emptyset$; see Figure 4.9.

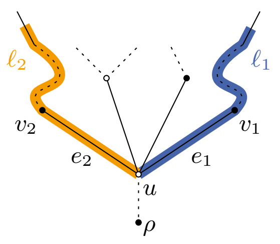
Next, we introduce a data structure that encodes for each edge (u, v) of T all maximal (u, v)-rooted curves as $O(n)$ superposition-free curves in $\mathcal{E}(T_u)$. In particular, each of those curves lies on a single road section such that all (u, v)-rooted curves ending on that curve are maximal and label the same number of road sections.

Definition 4.1 (Linearization). *Let $e = (u, v)$ be an edge of T. A tuple $(L, \overline{\omega})$ is called a* linearization *of e, if L is a set of superposition-free curves and $\overline{\omega}\colon L \longrightarrow \mathbb{R}$ such that*
(1) for each curve $c \in L$ there is a road section e' in T_u with $c \subseteq \mathcal{E}(e')$,
(2) for each e-rooted curve ℓ there is a curve $c \in L$ with $\text{length}(\ell) + \mathrm{d}_u \in I(c)$,
(3) for each point p of each curve $c \in L$ there is a maximal e-rooted curve ℓ with $h(\ell) = p$ and $\overline{\omega}(c) = \omega(\mathcal{L}(\ell))$.

Assume that we apply Algorithm 1 on T' and that we currently consider the vertex u of T'. Hence, we can assume that for each vertex $v \neq u$ of T'_u its optimal candidate and



(a) Case 1: Regular vertex.



(b) Case 2: Junction vertex.

Figure 4.10: Application of linearizations. (a) The vertex u is a regular vertex. Hence, only vertical labels can end at u. (b) The vertex u is a junction vertex. Hence, only horizontal labels can cover u.

$\omega(\mathcal{L}(v))$ is given. We first explain how to speed up `OptCandidate` using linearizations. Afterwards, we present the construction of linearizations.

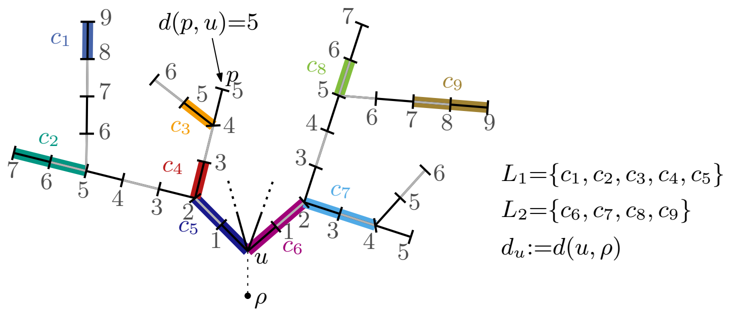
Application of Linearizations. Here we assume that the linearizations are given for the edges of T. Concerning the type of u we describe how to compute its optimal candidate.

Case 1, u is regular. If u is a leaf, the set $C(u)$ contains only $\perp_u$. Hence, assume that u has one outgoing edge $e = (u, v) \in E'$, which belongs to a road R. Let P be the longest path of vertices in T'_u that starts at u and does not contain any junction vertex. Note that the path must be unique. Further, by construction of T' the last vertex w of P must be a regular vertex in V, but not in $V' \setminus V$. We consider two cases; see Figure 4.10(a).

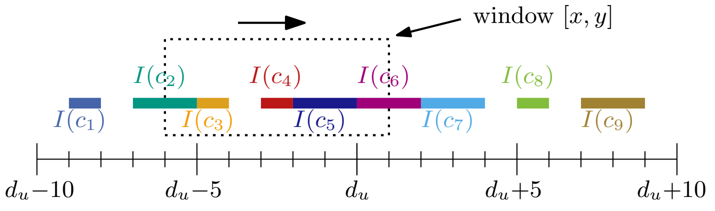
If $\mathrm{d}(u, w) \geq \lambda(R)$, the optimal candidate is either $\perp_u$ or the e-rooted curve ℓ of length $\lambda(R)$ that ends on $\mathcal{E}(P)$. By assumption and due to $\omega(\mathcal{L}(\perp_u)) = \omega(\mathcal{L}(v))$, we decide in $O(1)$ time whether $\omega(\mathcal{L}(\perp_u)) \geq \omega(\mathcal{L}(\ell))$, obtaining the optimal candidate.

If $\mathrm{d}(u, w) < \lambda(R)$, the optimal candidate is either $\perp_u$ or goes through a junction. Since w is regular, it has only one outgoing edge $e' = (w, x)$. Further, by the choice of P the edge e' is a junction edge in T; therefore the linearization $(L, \overline{\omega})$ of e' is given. In linear time we search for the curve $c \in L$ such that there is an e-rooted curve ℓ of length $\lambda(R)$ with its head on c. To that end we consider for each curve $c \in L$ its distance interval $I(c)$ and check whether there is $t \in I(c)$ with $t - \mathrm{d}_u = \lambda(R)$. Note that using a binary search tree for finding c speeds this procedure up to $O(\log n)$ time, however, this does not asymptotically improve the total running time. The e-rooted curve ℓ then can be easily constructed in $O(n)$ time by walking from c to u in $\mathcal{E}(T)$.

If such a curve c exist, by definition of a linearization the optimal candidate is either $\perp_u$ or ℓ, which we can decide in $O(1)$ time by checking $\omega(\mathcal{L}(\perp_u)) \geq \omega(\mathcal{L}(\ell))$. Note that we have $\omega(\mathcal{L}(\perp_u)) = \omega(\mathcal{L}(v))$ and $\omega(\mathcal{L}(\ell)) = \overline{\omega}(c)$. If c does not exist, again by



(a) Tree representation.



(b) Interval representation.

Figure 4.11: Constructing the optimal candidate of u based on the linearizations $(L_1, \overline{\omega}_1)$ and $(L_2, \overline{\omega}_2)$. The tree is annotated with distance marks.

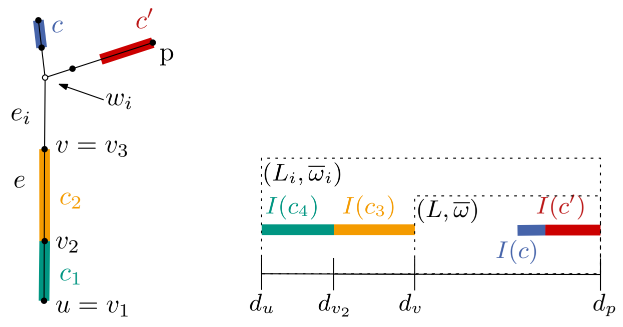
definition of a linearization there is no vertical label $\ell \in C(u)$ and $\perp_u$ is the optimal candidate.

Case 2, u is a junction vertex. The set $C(u)$ contains horizontal labels. Let ℓ be such a label and let $e_1 = (u, v_1)$ and $e_2 = (u, v_2)$ be two junction edges in E covered by ℓ; see Figure 4.10(b). Then there is an e_1-rooted curve ℓ_1 and an e_2-rooted curve ℓ_2 whose composition is ℓ. Further, we have $\omega(\mathcal{L}(\ell)) = \omega(\mathcal{L}(\ell_1) \cup \mathcal{L}(\ell_2)) + \sum_{v \in N(u) \setminus \{v_1, v_2\}} \omega(\mathcal{L}(v))$. We use this as follows.

Let e_1 and e_2 be two outgoing edges of u that belong to the same road R, and let $(L_1, \overline{\omega}_1)$ and $(L_2, \overline{\omega}_2)$ be the linearizations of e_1 and e_2, respectively. We define for e_1 and e_2 and their linearizations the operation opt-cand(L_1, L_2) that finds an optimal candidate of u restricted to labels covering e_1 and e_2.

For $i = 1, 2$ let $d_i = \max\{\mathrm{d}_u \mid u \text{ is vertex of } T_{v_i}\}$ and let $f_u(t) = \mathrm{d}_u - (t - \mathrm{d}_u) = 2\,\mathrm{d}_u - t$ be the function that “mirrors” the point $t \in \mathbb{R}^2$ at d_u. Applying $f_u(t)$ on the boundaries of the distance intervals of the curves in L_1, we first mirror these intervals such that they are contained in the interval $[2\,\mathrm{d}_u - d_1, \mathrm{d}_u]$; see Figure 4.11. Thus, the curves in $L_1 \cup L_2$ are mutually superposition-free such that their distance intervals lie in $J = [2\,\mathrm{d}_u - d_1, d_2]$.

We call an interval $[x, y] \subseteq J$ a *window*, if it has length $\lambda(R)$, $\mathrm{d}_u \in [x, y]$ and there are curves $c_1 \in L_1$ and $c_2 \in L_2$ with $x \in I(c_1)$ and $y \in I(c_2)$; see Figure 4.11. By the definition of a linearization there is a maximal e_1-rooted curve ℓ_1 ending on c_1 and



(a) Tree representation. **(b)** Interval representation.

Figure 4.12: 1st step of constructing a linearization: For each edge e_i its linearization $(L, \overline{\omega})$ is extended to a linearization $(L_i, \overline{\omega}_i)$ of T_i.

a maximal e_2-rooted curve ℓ_2 ending on c_2 such that length(ℓ_1) + length$(\ell_2) = \lambda(R)$. Consequently, the composition of ℓ_1 and ℓ_2 forms a horizontal label ℓ with $\omega(\mathcal{L}(\ell)) = \omega(\mathcal{L}(\ell_1) \cup \mathcal{L}(\ell_2)) + \sum_{v \in N(u) \setminus \{v_1, v_2\}} \mathcal{L}(v)$; we call $\omega(\mathcal{L}(\ell))$ the *value* of the window. Using a simple sweep from left to right we compute for the distance interval $I(c)$ of each curve $c \in L_1 \cup L_2$ a window $[x, y]$ that starts or ends in $I(c)$ (if such a window exists). The result of opt-cand(L_1, L_2) is then the label ℓ of the window with maximum value. For each pair e_1 and e_2 of outgoing edges we apply opt-cand(L_1, L_2) computing a label ℓ. By construction either the label ℓ with maximum $\omega(\ell)$ or $\perp_u$ is the optimal candidate for u, which we can check in $O(1)$ time. Later on we prove that we consider only linearizations of linear size. Since each vertex of T' has constant degree, we obtain the next lemma.

Lemma 4.3. *For each $u \in V'$ the optimal candidate can be found in $O(n)$ time.*

Construction of Linearizations. We now show how to recursively construct a linearization for an edge $e = (u, v)$ of T. To that end we assume that we are given the subdivision tree T' of T and the linearizations for the outgoing edges $e_1 = (v, w_1), \dots, e_k = (v, w_k)$ of v that belong to the same road R as e. Further, we can assume that we have computed the weight $\omega(\mathcal{L}(w))$ for all vertices w in T'_u excluding u. In case that two vertices of those vertices share the same position in $\mathcal{E}(T'_u)$ we remove that one with less weight. Let T_i be the tree induced by the edges e, e_i and the edges of the subtree rooted at w_i. As a first step we compute for each linearization $(L, \overline{\omega})$ of each edge e_i a linearization $(L_i, \overline{\omega}_i)$ for e restricted to tree T_i, i.e., conceptually, we assume that T_u only consists of T_i's edges.
If e is a junction edge we set $L_i \leftarrow L$ and weight each curve $c \in L_i$ as follows.

$$\overline{\omega}_i(c) \leftarrow \overline{\omega}(c) + \sum_{w \in N(v) \setminus \{w_i\}} \omega(\mathcal{L}(w))$$

Otherwise, if e is a road section, let $v_1, \dots, v_l$ be the vertices of the subdivision tree T' that lie on e, i.e., $\mathcal{E}(v_j) \in \mathcal{E}(e)$ for all $1 \le j \le l$; see Figure 4.12. We assume

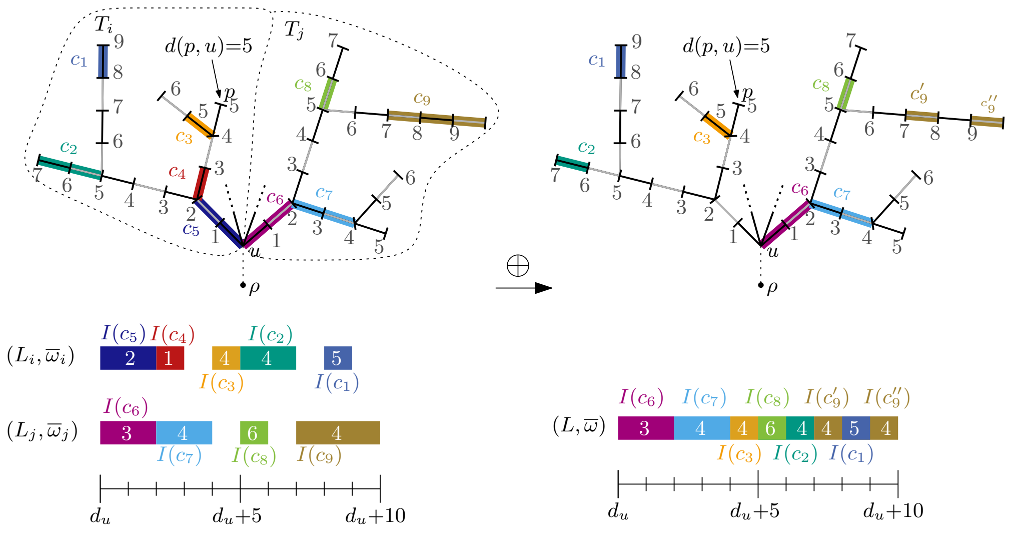


Figure 4.13: 2nd step of constructing a linearization: Merging the linearizations of the trees T_i and T_j.

that $\mathrm{d}(v_1) < \ldots < \mathrm{d}(v_l)$, which in particular yields $v_1 = u$ and $v_l = v$. Let c_1 be the curve $\mathcal{E}((v_1, v_2))$ and for $2 \leq j < l$ let c_j be the curve $\mathcal{E}((v_j, v_{j+1})) \setminus \mathcal{E}(v_j)$. Hence, we have $\bigcup_{j=1}^{l} c_j = \mathcal{E}(e)$ and $c_j \cap c_{j'} = \emptyset$ for $1 \leq j < j' < l$. We set

$$L_i \leftarrow L \cup \bigcup_{j=1}^{l-1} \{c_j\}$$

We weight each curve $c \in L_i$ as follows. If c is contained in L, we set

$$\overline{\omega}_i(c) \leftarrow \overline{\omega}(c) + 1$$

Otherwise, c is a sub-curve of $\mathcal{E}(e)$ and there exists a j with $c = c_j$. We set

$$\overline{\omega}_i(c) \leftarrow \omega(\mathcal{L}(v_{j+1}) \cup \{\ell_c\}),$$

where $\ell_c \subseteq \mathcal{E}(e)$ is an e-rooted curve that starts at $\mathcal{E}(u)$ and ends on c. The next lemma shows that this transformation yields a linearization as desired.

Lemma 4.4. *For each outgoing edge e_i with linearization $(L, \overline{\omega})$ the tuple $(L_i, \overline{\omega}_i)$ is a linearization of e restricted to the tree T_i.*

Proof. We use the same notation as used above.

First of all, the set L_i contains only curves that do not superpose each other: Since L contains only curves that do not superpose each other, the only curves that could superpose another curve in L_i are contained in $L_i \setminus L$. Since $L_i \setminus L$ is empty for a junction edge, we can assume that e is a road section. By construction those curves in $L_i \setminus L$ partition $\mathcal{E}(e)$ without intersecting each other. Further, by assumption no two road sections share a common vertex and since all curves of L are contained in $\mathcal{E}(T_v)$, the curves in $L_i \setminus L$ cannot superpose any curve in L.

We now prove that L_i satisfies the three conditions of a linearization. First assume that e is a road section.

Condition (1). Since L is a linearization, each curve of L must be a sub-curve of a road section. Further, the curves $L_i \setminus L$ are sub-curves of the road section e.

Condition (2). First consider an e-rooted curve ℓ that either ends on e_i or on an edge of T_{w_i}. Recall that $h(\ell)$ must lie on a road section. Then there is an e_i-rooted curve ℓ' with $\ell' \subseteq \ell$ and $h(\ell) = h(\ell')$. Hence, there is a curve $c \in L$ with $\text{length}(\ell') + \mathrm{d}_v \in I(c)$. Since ℓ' is a sub-curve of ℓ, we also have $\text{length}(\ell) + \mathrm{d}_u \in I(c)$. Now, consider an e-rooted curve ℓ that ends on e, then obviously by construction there is a curve $c \in L_i \setminus L$ with $\text{length}(\ell) + \mathrm{d}_u \in I(c)$.

Condition (3). First consider an arbitrary curve $c \in L_i \setminus L$ and let ℓ be any e-rooted curve that ends on c. Further, let $v_1, \ldots, v_l$ be the vertices of the subdivision tree T' that lie on e as defined above. By construction there is an edge (v_j, v_{j+1}) with $1 \leq j < l$ and $c \subseteq \mathcal{E}(v_j, v_{j+1})$. It holds

$$\omega(\mathcal{L}(\ell)) = \omega(\mathcal{L}(v_{j+1}) \cup \{\ell\}) = \overline{\omega}_i(c)$$

Obviously, ℓ must be maximal, because there is no other point in $\mathcal{E}(T_i)$ having the same distance to ρ as $h(\ell)$ has.

Finally, consider a curve $c \in L$ and let ℓ be any e-rooted curve that ends on c. As L is a linearization of e_i, for each point p on c there must be an e_i-rooted curve ℓ' with $h(\ell') \in c$. We choose ℓ' such that $h(\ell') = h(\ell)$. Since ℓ' is a maximal e_i-rooted curve, the curve ℓ must be a maximal e-rooted curve. Further, ℓ labels one road section more than ℓ'. Hence, we obtain

$$\omega(\mathcal{L}(\ell)) = \omega(\mathcal{L}(\ell')) + 1 = \overline{\omega}(c) + 1 = \overline{\omega}_i(c)$$

Now consider the case that e is a junction edge. *Condition (1)* and *Condition (2)* follow by the same arguments as stated above with the simplification that $L_i = L$.

Condition (3). Let c be a curve in L_i and let ℓ be any e-rooted curve that ends on c. Further, let ℓ' be the e_i-rooted sub-curve of ℓ that starts at $\mathcal{E}(v)$ and ends at $h(\ell)$; by definition of L such a curve exists. It holds

$$\omega(\mathcal{L}(\ell)) = \omega(\mathcal{L}(\ell')) + \sum_{w \in N(v) \setminus \{w_i\}} \omega(\mathcal{L}(w)) = \overline{\omega}(c) + \sum_{w \in N(v) \setminus \{w_i\}} \omega(\mathcal{L}(w)) = \overline{\omega}_i(c)$$

Since ℓ' is a maximal e_i-rooted curve, it directly follows that ℓ is a maximal e-rooted curve with respect to T_i. □

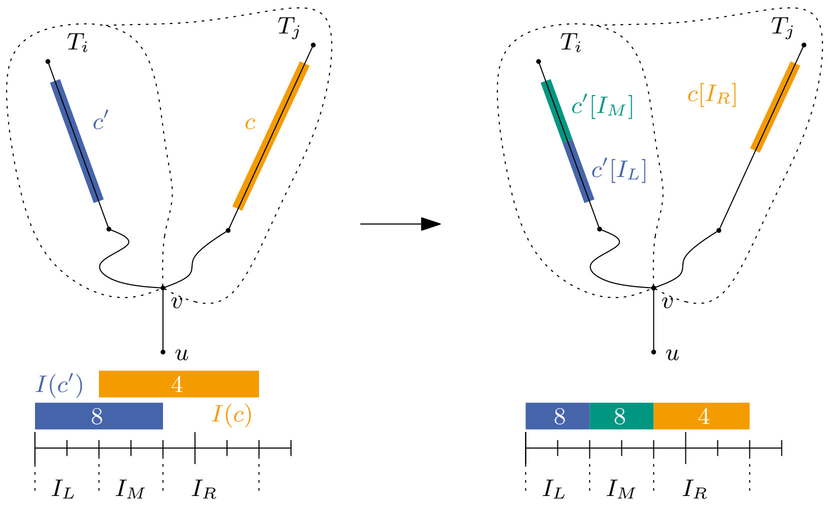


Figure 4.14: Illustration of merging two linearizations $(L_i, \overline{\omega}_i)$ and $(L_j, \overline{\omega}_j)$ into one linearization $(L_1, \overline{\omega}_i)$. The trees are annotated with distance marks.

In the next step we define an operation $\oplus$ by means of which two linearizations $(L_i, \overline{\omega}_i)$ and $(L_j, \overline{\omega}_j)$ can be combined to one linearization $(L_i, \overline{\omega}_i) \oplus (L_j, \overline{\omega}_j)$ of e that is restricted to the subtree $T_{i,j}$ induced by the edges of T_i and T_j. Consequently, $\bigoplus_{i=1}^{k}(L_i, \overline{\omega}_i)$ is the linearization of e without any restrictions.

We define $(L, \overline{\omega}) = (L_i, \overline{\omega}_i) \oplus (L_j, \overline{\omega}_j)$ as follows; for illustration see also Figure 4.14. Let $c_1, \dots, c_\ell$ be the curves of $L_i \cup L_j$ such that for any two curves c_s, c_t with $s < t$ the left endpoint of $I(c_s)$ lies to the left of the left endpoint of $I(c_t)$; ties are broken arbitrarily. We successively add the curves to L in the given order enforcing that the curves in L remain superposition-free. Let c be the next curve to be added to L.

Without loss of generality, let $c \in L_i$. The opposite case can be handled analogously. In case that there is no curve superposing c, we add c to L and set $\overline{\omega}(c) = \overline{\omega}_i(c)$. If c superposes a curve in L, due the order of insertion, there can only be one curve c' in L that superposes c. First we remove c' from L. Let I_M be the interval describing the set $I(c) \cap I(c')$, and let I_L and I_R be the intervals describing the set $I(c) \cup I(c') \setminus (I(c) \cap I(c'))$ such that I_L lies to the left of I_M and I_R lies to the right of I_M.

We now define three curves c_L, c_M and c_R with $I(c_L) = I_L$, $I(c_M) = I_M$ and $I(c_R) = I_R$ such that each of these three curves is a sub-curve of either c or c'. To that end let $c[I]$ denote the sub-curve of c whose distance interval is I. We define the curve c_R with weight $\overline{\omega}(c_R)$ as

$$(c_R, \overline{\omega}(c_R)) = \begin{cases} (c[I_R], \overline{\omega}_i(c)), & \text{if } I_R \subseteq I(c) \\ (c'[I_R], \overline{\omega}(c')), & \text{if } I_R \subseteq I(c') \end{cases}$$

The curve c_L and its weight $\overline{\omega}(c_L)$ is defined analogously.

The curve c_M and its weight $\overline{\omega}(c_M)$ is

$$(c_M, \overline{\omega}(c_M)) = \begin{cases} (c[I_M], \overline{\omega}_i(c)), & \text{if } \overline{\omega}_i(c) \geq \overline{\omega}(c') \\ (c'[I_M], \overline{\omega}(c')), & \text{if } \overline{\omega}_i(c) < \overline{\omega}(c') \end{cases}$$

The next lemma proves that $(L_i, \overline{\omega}_i) \oplus (L_j, \overline{\omega}_j)$ is a restricted linearization.

Lemma 4.5. *Let $(L_i, \overline{\omega}_i)$ and $(L_j, \overline{\omega}_j)$ be two linearizations of $e = (u, v)$ that are restricted to the trees T_i and T_j, respectively. Then $(L, \overline{\omega}) = (L_i, \overline{\omega}_i) \oplus (L_j, \overline{\omega}_j)$ is a linearization of e restricted to $T_{i,j}$. The operation needs $O(|L_i| + |L_j|)$ time.*

Proof. First of all, the set L contains only curves that are pairwise free from any superpositions. This directly follows from the construction that curves c and c' superposing each other are replaced by three superposition-free curves c_L, c_M and c_R. Due to $I(c_L) \cup I(c_M) \cup I(c_R) = I(c) \cup I(c')$ the first and second condition of a linearization is satisfied.

We finally prove that Condition (3) of a linearization is satisfied by doing an induction over the curves inserted to L. Let L^k be L after the k-th insertion step. Since L^0 is empty, the condition obviously holds for L^0. So assume that we insert c to L^k obtaining the set L^{k+1}. Without loss of generality assume that $c \in L_i$. If c does not superpose any curve in L^k, the condition directly follows from the definition of c. So assume that $c' \in L^k$ superposes c. Since $c \in L_i$, the curve c' is contained in $\mathcal{E}(T_j)$. We remove c' from L^k and insert the curves c_R, c_M and c_L as defined above. We prove that all three curves satisfy Condition (3).

Consider in the following the subtree $T_{i,j}$ of T_u restricted to the edges of T_i and T_j. We set $c_R = c[I_R]$ and set $\overline{\omega}(c_R) = \overline{\omega}_i(c)$, if $I_R \subseteq I(c)$. In that case there is no e-rooted curve $\ell \subseteq \mathcal{E}(T_j)$ with $\text{length}(\ell) + \text{d}_u \in I_R$, i.e., either there is no curve ℓ in $\mathcal{E}(T_j)$ with $t(\ell) = \mathcal{E}(u)$ and $\text{length}(\ell) + \text{d}_u \in I_R$, or any curve in $\mathcal{E}(T_j)$ with $t(\ell) = \mathcal{E}(u)$ and $\text{length}(\ell) + \text{d}_u \in I_R$ ends on a junction edge. Consequently, any e-rooted curve ℓ with $\text{length}(\ell) + \text{d}_u \in I_R$ and in particular any maximal e-rooted curve ℓ with $\text{length}(\ell) + \text{d}_u \in I_R$ lies in $\mathcal{E}(T_i)$. Thus, the curve c_R satisfies Condition (3). For the case $I_R \subseteq I(c')$ and the curve c_L we can argue analogously.

So consider the curve c_M. Without loss of generality we assume that $\overline{\omega}_i(c) \geq \overline{\omega}(c')$. The opposite case can be handled analogously. For any maximal e-rooted curve ℓ in $\mathcal{E}(T_j)$ with $\text{length}(\ell) + \text{d}_u \in I_M$ it must be true that $\omega(\mathcal{L}(\ell)) \leq \overline{\omega}(c_M)$. Further, since $c_M \subseteq c$ and c satisfies Condition (3) with respect to T_i, c_M satisfies the Condition (3) with respect to $T_{i,j}$. □

Lemma 4.4 and Lemma 4.5 yield that $\bigoplus_{i=1}^{k}(L_i, \omega_i)$ is the linearization of e without any restrictions. Computing it needs $O(\sum_{i=1}^{k} |L_i|)$ time.

Note that when computing optimal candidates (see *Application of linearizations*) we are only interested in e-rooted curves ℓ that have length at most $\lambda(R)$, where R is the road of e. Hence, when constructing $(L_i, \overline{\omega}_i)$ for an edge e_i in the first step, we discard any curve c of L_i that does not allow an e-rooted curve that both ends on c and has length at most $\lambda(R)$; the curve c is not necessary for our purposes. Hence, we conceptually restrict T_i to the edges that are reachable from u by one label length. It is not hard to see that T' restricted to $\mathcal{E}(T_i)$ contains only $O(n)$ vertices, because each vertex of $V' \setminus V$ is induced by a chain of tightly packed vertical labels, whereas each label has length $\lambda(R)$. Hence, T' restricted to $\mathcal{E}(T_i)$ contains for each such chain at most one vertex of $V' \setminus V$. Further, the endpoints of the curves in L_i are induced by the vertices of T'. Hence, by discarding the unnecessary curves of L_i the set L_i has size $O(n)$. Altogether, by Lemma 4.5 and due to the constant degree of each vertex we can construct $\bigoplus_{i=1}^{k}(L_i, \omega_i)$ in $O(\sum_{i=1}^{k} n) = O(n)$ time.

When constructing $\mathcal{L}(u)$ for u as described in Algorithm 1, we first build the linearization L_e of each of u's outgoing edges. By Lemma 4.3 we can find in $O(n)$ time the optimal candidate of u. Then, due to the previous reasoning, the linearization of an edge of T and the optimal candidate of a vertex u can be constructed in $O(n)$ time. Altogether we obtain the following result.

Proposition 4.1. *For a road map $\mathcal{M}$ with a tree T as underlying road graph, MaxLabeledRoads can be solved in $O(n^3)$ time.*

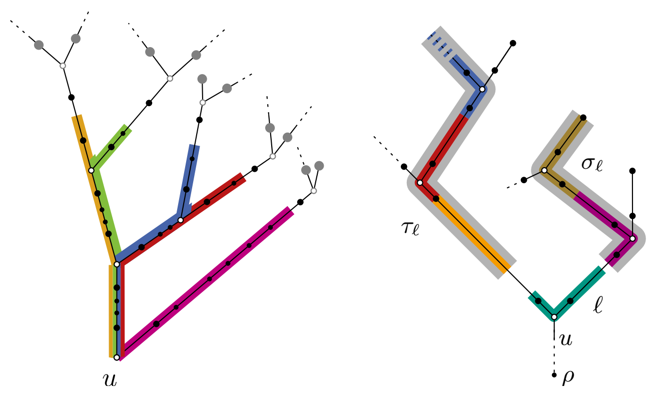
4.4.3 Improvements on Storage Consumption

Since T' contains $O(n^2)$ vertices, the algorithm needs $O(n^2)$ space. This can be improved to $O(n)$ space. To that end T' is constructed *on the fly* while executing Algorithm 1. Parts of T' that become unnecessary are discarded. We prove that it is sufficient to store $O(n)$ vertices of T' at any time such that the optimal labeling can still be constructed.

When constructing the optimal labeling of T, we build for each edge (u, v) of T its linearization based on the linearization of the outgoing edges of v. Afterwards we discard the linearizations of those outgoing edges. Since each vertex has constant degree, considering the vertices of T' in an appropriate order, it is sufficient to maintain a constant number of linearizations at any time.

Hence, because each linearization has size $O(n)$, we need $O(n)$ space for storing the required linearizations in total. However, we store for each vertex u of T' the weight $\omega(\mathcal{L}(u))$ and its optimal candidate. As T' has size $O(n^2)$ the space consumption is $O(n^2)$. In the following, we improve that bound to $O(n)$ space.
We call a vertex $v \in V'$ *reachable* from a vertex $u \in V'$, if there is a curve $\ell \subseteq \mathcal{E}(T'_u)$ that starts at $\mathcal{E}(u)$ and that is contained in the embedding of a road R with $\lambda(R) \geq$ length(ℓ) such that $\mathcal{E}(v) \in \ell$ or $v \in N(\ell)$, where length(ℓ) denotes the length of ℓ; see Figure 4.15(a). The set $\mathbb{R}_u$ contains all vertices of T'_u that are reachable from u. The next lemma shows that $\mathbb{R}_u$ has linear size.

Lemma 4.6. *For any vertex u of T' the set $\mathbb{R}_u$ has size $O(n)$.*



(a) Reachable vertices. **(b)** Chains of label ℓ.

Figure 4.15: Improvements on storage consumption. (a) Illustration of the reachable vertices from u. Vertices not reachable from u are marked gray. (b) Illustration for reconstructing the computed labeling.

Proof. Recall how T' is constructed: For each vertex $v \in V$ we construct a chain C of tightly packed vertical valid labels, which starts at $\mathcal{E}(v)$, is directed towards ρ, and ends when either the road ends, or adding the next label does not increase the number of labeled road sections. Each label of such a chain C induces one vertex of T'. Hence, C induces a set V_C of vertices in T'. We show that for each chain C the set $V_C \cap \mathbb{R}_u$ contains at most two vertices. As we construct n chains in order to build T' the claim follows.

For the sake of contradiction assume that there is a chain C and a vertex u in T' such that $V_C \cap \mathbb{R}_u$ contains more than two vertices. Without loss of generality we assume that $V_C \cap \mathbb{R}_u$ contains three vertices, which we denote by v_1, v_2 and v_3. We further assume that $\mathrm{d}_{v_1} < \mathrm{d}_{v_2} < \mathrm{d}_{v_3}$. By construction all labels in C lie in the embedding of the same road R_C, and $\mathrm{d}(v_1, v_2) \geq \lambda(R_C)$ and $\mathrm{d}(v_2, v_3) \geq \lambda(R_C)$. By definition of C there is a vertical curve $\ell \in \mathcal{E}(T'_u)$ that starts at $\mathcal{E}(u)$ and contains v_1, v_2 and v_3. Let e be the outgoing edge of u in T' whose embedding is covered by ℓ and consider the sub-curve $\ell' \subseteq \ell$ with length $\lambda(R_C)$ that starts at u. By definition of $\mathbb{R}_u$, we know for each v_i with $1 \leq i \leq 3$ that either its embedding is contained in ℓ' or $v_i \in N(\ell')$. From the definition of $N(\ell')$ and the fact that all three vertices lie on ℓ, it directly follows that only v_3 may be contained in $N(\ell')$. Hence, $\mathcal{E}(v_1), \mathcal{E}(v_2) \in \ell'$. Further, because $v_2 \notin N(\ell')$, we have $\mathcal{E}(v_2) \neq h(\ell')$, which implies $\mathrm{d}(v_1, v_2) < \lambda(R)$ and contradicts $\mathrm{d}(v_1, v_2) \geq \lambda(R)$. □

Assume that we apply Algorithm 1 considering the vertex u. When constructing u's optimal candidate, by Lemma 4.6 it is sufficient to consider the vertices of T'_u that lie in $\mathbb{R}_u$. On that account we discard all vertices of T'_u that lie in $V' \setminus V$, but not in $\mathbb{R}_u$. Further, we compute the vertices of $V' \setminus V$ that subdivide the incoming edge $(t, u) \in E$ *on demand*, i.e., we compute them, when constructing the optimal candidate of t. Hence, we have linear space consumption.
However, when discarding vertices of T', we lose the possibility of reconstructing the

labeling. We therefore annotate each vertex $u \in V$ of the original tree T with further information. To that end consider a canonical labeling $\mathcal{L}$ of T. Let ℓ be a horizontal label of $\mathcal{L}$ and let e be the edge of T on which ℓ's head is located; see Figure 4.15(b). Either, no other label of $\mathcal{L}$ ends on e, or another label ℓ' ends on e that belongs to a chain σ_ℓ of tightly packed vertical labels. Analogously, we can define the chain τ_ℓ with respect to edge e' on which ℓ's tail is located. On that account we store for a junction vertex $u \in V$ not only its optimal candidate $\ell \in C(u)$, but also the two chains σ_ℓ and τ_ℓ, if they exist. Note that such a chain of tightly packed vertical labels is uniquely defined by its start and endpoint, which implies that $O(1)$ space is sufficient to store both chains. Using a breadth-first search we can easily reconstruct those chains in linear time. For a regular vertex $u \in V$ we analogously store σ_ℓ of its optimal candidate $\ell \in C(u)$, if it exists. Since ℓ is vertical, we do not need to consider its tail. For the special case that $\ell = \perp_u$, we define that σ_ℓ is the chain of tightly packed vertical labels that ends on the only outgoing edge e of u. Summarizing, the additional information together with the optimal candidates stored at the vertices $u \in V$ of the original tree are sufficient to reconstruct the labeling of T. Together with Proposition 4.1 we obtain the following result.

Theorem 4.4. *For a road map $\mathcal{M}$ with a tree T as underlying road graph, MaxLabeledRoads can be solved in $O(n^3)$ time using $O(n)$ space.*

4.5 Conclusions

In this chapter, we investigated the problem of maximizing the number of labeled road sections in a labeling of a road map; we showed that it is NP-hard in general, but can be solved in $O(n^3)$ time and linear space for the special case of trees.

The underlying road graphs of real-world road maps are rarely trees. However, in the next chapter, we show that road maps can be decomposed into a large number of subgraphs by placing trivially optimal road labels and removing the corresponding edges from the graph. It turns out that a vast majority of the resulting subgraphs are actually trees, which we can label optimally by our proposed algorithm. As a consequence, this means that a large fraction of all road sections in our real-world road graphs can be labeled optimally by combining this simple preprocessing strategy with the tree labeling algorithm. We further investigate heuristic and exact approaches for labeling the remaining non-tree subgraphs (e.g., by finding suitable spanning trees and forests).

5 Label Placement in Road Maps: An Algorithmic Framework

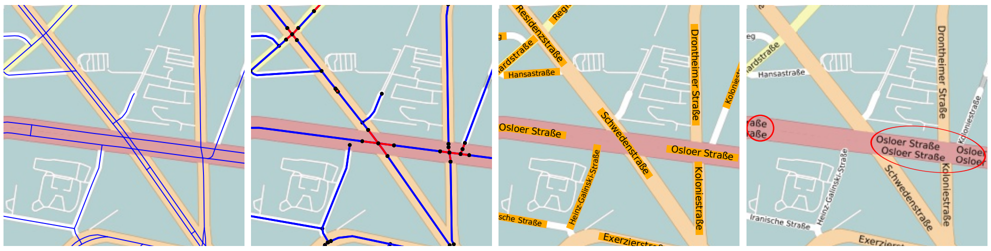
Abstract. In the previous chapter, we presented a general model for label placement in road maps. However, without any adaptions this model is of theoretical nature and cannot be directly applied in practice. In this chapter, we explain how to decompose the road network into logically coherent road sections, e.g., parts of roads between two junctions. Based on this decomposition, we present and implement a new and versatile framework for placing labels in road maps. In particular, we show that our tree-based algorithm of the previous chapter can in fact be used successfully as the core of an efficient and practical road labeling algorithm. In an experimental evaluation with road maps of eleven major cities we show that our proposed labeling algorithm is both fast in practice and that it reaches near-optimal solution quality, where optimal solutions are obtained by mixed-integer linear programming. In comparison to the standard OpenStreetMap rendering engine Mapnik, our algorithm labels 31% more road sections on average.

This chapter is based on and partly taken from joint work with Martin Nöllenburg [NN16].

5.1 Introduction

At any given scale, road networks are typically drawn on maps as follows. Each road or road lane is represented as a thick, polygonal curve, i.e., a polygonal curve with non-zero width; see the background of Figure 5.1(a). If two (or more) such curves intersect, they form junctions. If two or more lanes of the same road closely run in parallel, they merge to one even thicker curve such that individual lanes become indistinguishable. We then want to place road labels inside these thick curves. More precisely, a *road label* can again be represented as a thick curve (the bounding shape of the road name) that is contained in and parallel to the thick curve representing its road; see Figure 5.1(c).

In contrast, the abstract road graph model of Chapter 4 used a simplified representation, which represents the road network and its labels as thin curves instead. More precisely, a road network is modeled as a planar embedded *abstract road graph* whose edges correspond to the skeleton of the actual thick curves. In this model a label is again a thin curve of certain length that is contained in the skeleton. Following the cartographic quality criteria (C1)–(C3) from Chapter 4, we place labels, i.e., find sub-curves of the skeleton, such that (1) each label starts and ends on road sections,



(a) OSM. **(b)** Graph G. **(c)** Labeling. **(d)** Mapnik.

Figure 5.1: The presented workflow. (a) The road network given by polylines (blue segments). (b) Phase 1: A graph G is created whose embedding is the simplified road network; blue segments: road sections, red segments: junction edges. (c) Phase 2: Creating the labeling using G. (d) A labeling produced by the OSM rendering engine Mapnik. The six labels of the road *Osloer Straße* are enclosed by red ellipses.

but not on junctions, (2) no two labels overlap, and (3) a maximum number of road sections are labeled. From a labeling of the abstract road graph it is straight-forward to transform each label back into its *text representation* by placing the individual letters of each label along the thick curves; see Figure 5.2(a).

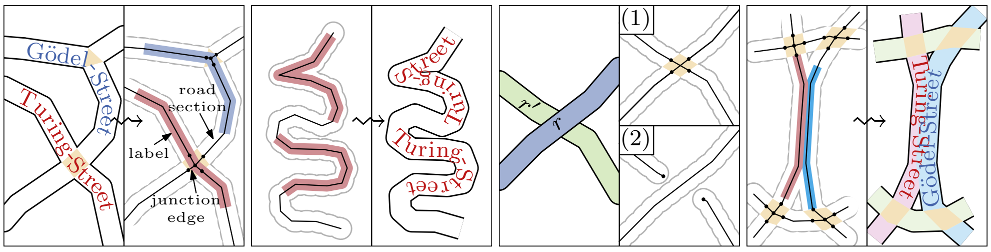
While this abstract road graph model allows theoretical insights (see Chapter 4), we cannot directly apply it to real-world road networks. Due to the following issues, we need to invest some effort in a preprocessing phase to guarantee that the resulting labels in the text representation do not overlap, look nicely, and are embedded in the roads' shapes.

Issue 1. If lanes run closely in parallel, their drawings in the road network merge to one thick curve and individual lanes become indistinguishable. Hence, in our abstract model, such lanes should be aggregated to a single road section that represents the skeleton of the merged curve, and labels should be contained in it; see Figure 5.1(b)–(c).

Issue 2. Real-world road networks are not planar, but edges may cross, namely at tunnels and bridges; see Figure 5.2(c). To avoid overlaps between labels placed on such road sections, we can either model the intersection as a regular junction of two roads or split one into two shorter road sections that do not cross the other road section. In both cases the road graph becomes planar. For our prototype we use the first variant (also used by Mapnik), because more road sections can be labeled.

Issue 3. In real-world road networks some road sections are possibly so long that the label should be repeated after appropriate distances.

Issue 4. Labels have a certain font size so that when transforming an abstract label curve into its text representation, labels of different roads may overlap due to their road sections being too close; see Figure 5.2(d).



(a) Road graph. **(b)** Curvy labels. **(c)** Crossings. **(d)** Retransformation.

Figure 5.2: Illustration of model and arising issues. (a) Sketch of a road network and its abstract road graph. (b) Labels are possibly curvy and have sharp bends making the text hardly legible. (c) Issue 2: Two ways to represent bridges and tunnels in the abstract road graph. (d) Issue 4: The text representation of labels may overlap, although the curve representation in the abstract road graph does not.

Contribution and Outline. In this chapter, we introduce a new, versatile algorithmic framework for placing non-overlapping labels in road networks that maximizes the number of labeled road sections. We keep the algorithmic components easily exchangeable. In Section 5.2 we discuss and expand the model introduced in Chapter 4. Afterwards, we present a workflow for labeling road networks consisting of two phases; see Figure 5.1. We now sketch both phases.

Phase 1 (Section 5.3). We translate the given road network into a semantic representation (an abstract road graph) that identifies pieces of the road network that belong semantically together. To that end, we simplify the road network, e.g., by merging lanes closely running in parallel. By design this simplification maintains the overall geometry of the road network and only merges structures in the data that should not be labeled independently. This phase resolves Issue 1–Issue 4. It is not part of the labeling optimization process.

Phase 2 (Section 5.4). Based on the abstract road graph, we create an actual labeling using one of three algorithms: a naive base-line algorithm, a heuristic extending our tree-based algorithm of Chapter 4 and a mixed-integer linear programming (MILP) formulation. To tweak those approaches, we may optionally decompose the graph into smaller pieces with the property that optimal labelings of the components yield an optimal labeling of the entire graph.

As proof of concept, we implemented the core of the framework only taking the most important cartographic criteria into account. However, with some engineering it can be easily enhanced to more complex models, e.g., enforcing minimum distances between labels, abbreviating road names, or using alternative definitions of road sections. In Section 5.5, we present a detailed evaluation of our framework on eleven sample city maps. Due to its availability and popularity in practice, we compare our results against the standard OpenStreetMap (OSM) render engine Mapnik as a representative of local heuristics; it uses a strategy similar to [Chi00, Str01]. We show that our tree-based algorithm is fast and yields near-optimal labelings that improve upon Mapnik. We provide interactive maps illustrating our labelings online: `http://i11www.iti.kit.edu/roadlabeling/`.

5.2 Adaptions on the Abstract Road Graph Model.

We have introduced the abstract road graph in Chapter 4, but in this chapter we take a slightly different view on the model. We therefore repeat the definition of the model here and adapt the applied notation, see also Figure 5.1(b) and Figure 5.2(a). A road network (in an abstract sense) is a planar geometric *graph* $G = (V, E)$, where each vertex $v \in V$ has a position in the plane and each edge $\{u, v\} \in E$ is represented by a polyline whose end points are u and v. Each edge further has a *road name*. A maximal connected subgraph of G consisting of edges with the same name forms a *road* R. The length of the name of R is denoted by $\lambda(R)$. Each edge $e \in E$ is either a *road section*, i.e., the part of a road in between two junctions, or a *junction edge*, which models road junctions. Formally, a *junction* is a maximal connected subgraph of G that only consists of junction edges. We require that no two road sections in G are incident to the same vertex and that vertices incident to road sections have at most degree 2. Thus, the road graph G decomposes into road sections, separated by junctions.

We say a point p lies on G, if there is an edge $e \in E$ whose polyline contains p. Hence, a polyline ℓ (in particular a single line segment) lies on G if each point of ℓ lies on G. Further, ℓ *covers* e, if there is a point of ℓ that lies on e. If each point of e is covered by ℓ, e is *completely covered*. The *geodesic distance* of two points on G is the length of the shortest polyline on G connecting both points.

A *label* of a road R is a simple open polyline ℓ on G that has length $\text{length}(\ell) = \lambda(R)$, ends on road sections of G, and whose segments only lie on edges of R. The start point of ℓ is denoted as the *head* $h(\ell)$ and the endpoint as the *tail* $t(\ell)$. Obviously, the edges that are covered by ℓ form a path $\mathcal{P}_\ell = (e_1, e_2, \cdots, e_{k-1}, e_k)$ of edges such that e_1, and e_k are (partly) covered and $e_2, \ldots, e_{k-1}$ are completely covered by ℓ. If e_i is a road section (and not a junction edge), we say that e_i is *labeled* by ℓ.

We extend the above abstract road graph model and restrict ourselves to *well-shaped* labels, i.e., labels that are not too curvy or do not contain broken type setting due to sharp bends; see Figure 5.2(b). Similar to Schwartges et al. [SWH14], we apply a local criterion to decide whether a label is well-shaped. To that end, we define a label ℓ to be *well-shaped* if for each covered edge $e \in \mathcal{P}_\ell$ there is a *well-shaped* piece of e that completely contains the part of ℓ on e. Further, we require that for each pair of incident edges of $\mathcal{P}_\ell$ the bend angle is at most $\alpha_{\max}$, where $\alpha_{\max}$ is a pre-defined constant. We redefine a *labeling* $\mathcal{L}$ to be a set of mutually non-overlapping, well-shaped labels. Our theoretic results of Chapter 4 remain valid for this restriction. In particular, only few minor technical adaptions are required for the tree labeling algorithm.

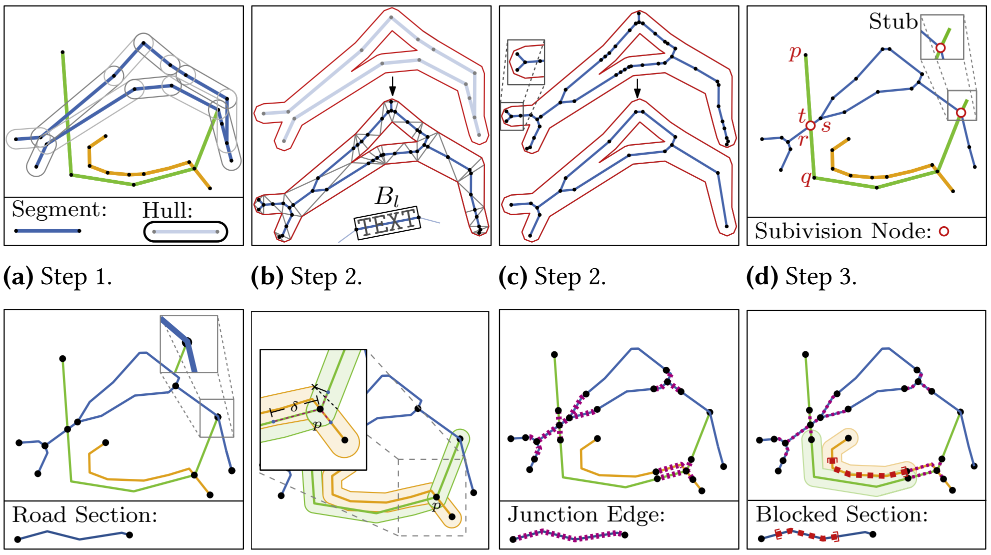
In order to identify *well-shaped* pieces of a polyline P with edges $e_1, \ldots, e_k$, we extend the approach presented by Schwartges et al. [SWH14]. They define the curviness $w(P)$ of P by summing up the bend angles α_i of all incident edge pairs e_i, e_{i+1}, i.e., $w(P) = \sum_{i=1}^{k-1} |\alpha_i|$ to determine the best label positions for any given label. We want to locally classify road pieces as well-shaped instead and adapt their idea as follows. Let S be a maximal sub-polyline of P with the property that any sub-polyline of S with length at most $l_{\max}$ has curviness at most $\alpha_{\max}$. Each such sub-polyline S forms a well-shaped piece of P and they can all be computed in $O(k)$ time. This local criterion for well-shapedness is based on the curviness of a fixed-width window sliding along the polyline; it is independent of the label length (similarly to what Mapnik does). In our experiments we set $l_{\max}$ to twice the length of the letter W and $\alpha_{\max} = 22.5°$, analogously to the parameters that Mapnik uses.

A *labeling* $\mathcal{L}$ for a road network is a set of mutually non-overlapping, well-shaped labels, where two labels ℓ and ℓ' *overlap* if they intersect in a point that is not their respective head or tail. Following the criteria (C1)–(C3) (see Chapter 4), the problem MaxLabeledRoads is to find a labeling $\mathcal{L}$ that labels a maximum number of road sections, i.e., no other labeling labels more road sections. In Chapter 4 we showed that MaxLabeledRoads is NP-hard in general, but can be solved in $O(|V|^3)$ time if G is a tree.

5.3 Phase 1 – Construction of Abstract Road Graphs

The first phase of our framework consists of transforming the input road network data into an abstract road graph while resolving the four issues mentioned in Section 5.1. Typically, road networks are given as a set of polylines that describe the roads and road lanes. Individual polylines do not necessarily form semantic components such as road sections. So as a first step, we break all polylines down into individual line segments (whose union forms the road network). Let L be the set of all these line segments. We further require that each line segment $l \in L$ is annotated with its *road name* $\mathrm{rn}(l)$, the stroke width $\mathrm{st}(l)$ and the color $\mathrm{co}(l)$ that are used to draw l, and finally the *font size* $\mathrm{fs}(l)$ that shall be used to display the name. We say that two line segments $l, l' \in L$ are *equally represented* if $\mathrm{st}(l) = \mathrm{st}(l')$ and $\mathrm{co}(l) = \mathrm{co}(l')$. We assume that $\mathrm{fs}(l) < \mathrm{st}(l)$ for any l; otherwise we set $\mathrm{st}(l) := \mathrm{fs}(l)$.

The workflow consists of the following five steps; see Figure 5.3. (1) Identification. Identify single *road components*, i.e., sets of line segments in the road network data that have the same name, are equally represented, and form a connected component. (2) Simplification. Simplify each road component such that lanes running closely in parallel are aggregated. (3) Planarization. Replace bridges and tunnels by artificial



(a) Step 1. **(b)** Step 2. **(c)** Step 2. **(d)** Step 3.

(e) Step 4. **(f)** Step 4. **(g)** Step 4. **(h)** Step 5.

Figure 5.3: Illustration of the steps applied in Phase 1. Segments of the same color have the same road name. For more details see the description of Phase 1.

junctions. (4) Transformation. Transform the segment representation into an abstract road graph. (5) Resolving Overlaps. Identify mutual overlaps of road sections and block them for label placement.

Below we describe each step in more detail. We define the *hull* of a line segment $l \in L$ to be the region of points whose Euclidean distance to l is at most $\mathrm{st}(l)$; see Figure 5.3(a). The hull of a polyline is then the union of its segments' hulls. We approximate hulls by simple polygons.

Step 1 – Identification. For each road name n, each color c and each font size f we define the intersection graph of the hulls of the line segments $L_{n,c,f} = \{l \in L \mid \mathrm{rn}(l) = n,\ \mathrm{co}(l) = c \text{ and } \mathrm{fs}(l) = f\}$. In this intersection graph each hull is a vertex and two vertices are connected if and only if the corresponding hulls intersect. In each (non-empty) intersection graph we identify all connected components, which we call *road components*; e.g., in Figure 5.3(a) the blue segments form a road component. Thus, based on L we obtain a set $\mathcal{C}$ of road components. By definition, each component $C \in \mathcal{C}$ has a unique name $\mathrm{rn}(C)$, stroke width $\mathrm{st}(C)$, color $\mathrm{co}(C)$ and font size $\mathrm{fs}(C)$.

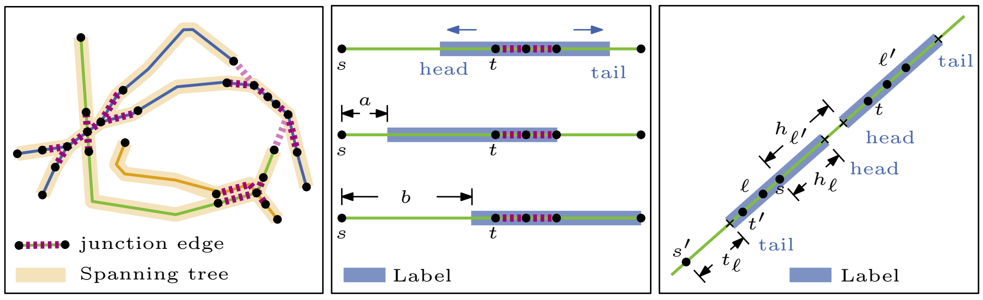
Step 2 – Simplification. For each road component $C \in \mathcal{C}$ we geometrically form the union of the corresponding hulls. Thus, the result is a simple polygon P (possibly with holes); see Figure 5.3(b), top. This polygon describes the contour of the road component as drawn on the map. We discard all polygons whose area is smaller than some threshold as they are too small to be labeled; we use the area of the letter W as threshold. For each remaining polygon P we construct the *skeleton* of P as a linear representation of the corresponding road component such that labels centered on the skeleton are guaranteed to be contained in P. This skeleton is computed based on the conforming Delaunay triangulation of the interior of P following Bader and Weibel [BW97]. For triangles that have one or three *internal* edges, i.e., edges that do not belong to the boundary of P, we connect the triangle centroid to the midpoints of the internal edges. For triangles with two internal edges, we simply connect the midpoints of these two edges, see Figure 5.3(b), bottom. From those line segments, we form a set of maximal polylines by appending all those line segments that meet at the midpoint of a triangle edge (but not at a triangle centroid). Since these polylines may consist of many vertices and meander *locally*, we simplify them using the Douglas-Peucker algorithm, but only if the simplified shortcuts keep a distance of at least $\mathrm{fs}(C)/2$ to the boundary of P, see Figure 5.3(c). Finally, we delete any segment l whose text box B_l is not completely contained in P. Here the *text box* B_l of l is defined as a rectangle centered at l with two sides parallel to l. These parallel sides have the same length as l, the two orthogonal sides have length $\mathrm{fs}(C)$, see Figure 5.3(b), bottom. Segments with the text box not contained in P may occur at the protrusions of the component where circular arcs are approximated by polylines, see Figure 5.3(c), top left. The remaining set of polylines forms the skeleton of P.

Thus, for each road component C we obtain a skeleton such that all text boxes of the skeleton edges are contained in P. This resolves Issue 1. We annotate each skeleton edge with the name, stroke width, color and font size of C.

Step 3 – Planarization. So far polylines of different road components may intersect at other points than their end points, e.g., polylines representing bridges and tunnels may cross other polylines. As motivated in Section 5.1, we subdivide these polylines to resolve intersections; see Figure 5.3(d). More precisely, if two line segments $\overline{pq}$ and $\overline{rs}$ of two polylines intersect at a point t, we replace them by the four segments $\overline{pt}$, $\overline{tq}$, $\overline{rt}$ and $\overline{ts}$. We do the intersection tests with a certain tolerance to identify T-crossings safely. However, this may yield short stubs that protrude junctions slightly; we remove these stubs. Thus, this step resolves Issue 2 and yields a set of annotated polylines only intersecting in vertices.

Step 4 – Transformation. Next we create the abstract road graph from the polylines of the previous step. As a result of Step 3, we know that any two polylines intersect only in vertices. We first take the union of all polylines, identify vertices that are common to two or more polylines and mark these vertices as *junction seeds*. This induces already a planar graph $G = (V, E)$ with polyline edges whose vertices V are either junction seeds or have degree 1. It remains to partition the edges of G into road sections and junction edges. Initially, we mark all edges as road sections. We distinguish two types of junction seeds in G.

If a junction seed v has degree at least 3, only two of its incident edges e and e'



(a) Tree **(b)** Milp: Equivalence **(c)** Milp: Variables

Figure 5.4: Illustration of the algorithms. Edges of the same color belong to the same road.

belong to the same road R and all other incident edges belong to different roads (and have a different road type than R) then we do not create any junction edges at v, see Figure 5.3(e), small box. Since R is the only road that may use the junction at v and it is visually clear that all other roads end at v, we can safely treat v as an internal vertex of a road section of R. So we disconnect all incident edges of v except e and e' from v and let each of them end at its own slightly displaced copy of v. The edges e and e' are merged at v and the new edge remains a road section. This resolves the situation as desired.

For all other junction seeds we create junction edges as follows. Let v be a junction seed and let E_v be the set of edges incident to v. We intersect the hulls of all edges in E_v and project their intersection points onto the corresponding edges, see Figure 5.3(f). For each edge $e \in E_v$ we determine the projection point p_e that is farthest away from v (in geodesic distance). If the distance between p_e and v exceeds a given threshold δ, we shift p_e to the point on e that has distance δ from v. Now we subdivide e at p_e and mark the edge $\{v, p_e\}$ as a junction edge; the other edge at p_e (if non-empty) remains a road section. The threshold δ ensures that roads running closely in parallel are not completely marked as junction edges. Figure 5.3(g) shows the resulting abstract road graph.

To resolve Issue 3 we subdivide road sections whose length exceeds a certain threshold (in our experiment 350 pixels) by inserting a very short junction edge.

Step 5 – Resolving Overlaps. By Step 2 the hulls of edges that belong to the same road component do not overlap. However, if two sections of different roads run closely in parallel, their hulls (and hence their labels) may overlap; Figure 5.3(h). We identify overlaps of the hulls of non-incident edges in G and block the corresponding parts of the edge whose road is less important for placing labels; ties are broken arbitrarily. More complex approaches using road displacement could be applied, however, we have chosen a simple solution. By design hulls of incident edges may only overlap if both are junction edges; those overlaps are handled by the labeling algorithms; see Section 5.4. This resolves Issue 4.

5.4 Phase 2 – Label Placement in Road Graphs

In this section we present the four different methods for solving MaxLabeledRoads that we subsequently evaluate in our experiments in Section 5.5. Furthermore, we describe a technique for decomposing road graphs into several smaller, independent components that may speed up computations.

5.4.1 Labeling Methods

BaseLine. An obvious base-line heuristic to obtain lower bounds is to simply place a well-shaped label on each individual road section that is long enough to admit such a label without extending into any junctions. We use this approach to show that it is beneficial to position labels across junctions.

Mapnik. Mapnik (`http://mapnik.org`) is a standard open source rendering engine for OpenStreetMap that includes a road labeling algorithm. The algorithm iteratively labels so-called *ways*, which are polylines describing line features in OpenStreetMap. Along each way it places labels with a certain spacing and locally ensures that labels do not intersect already placed labels of other ways. It does not use any semantic structure from the road network (e.g., road sections), but relies on how the contributors of OpenStreetMap modeled single ways. We may run the rendering algorithm and extract all placed labels from its output.

Tree. The tree-based heuristic makes use of our proposed algorithm that optimally solves MaxLabeledRoads if G is a tree; see Chapter 4. We use that algorithm as a *black box*. If G is a tree, our heuristic optimally labels G. Otherwise it computes a spanning tree T on G using Kruskal's algorithm and computes an optimal labeling for T; see Figure 5.4(a). We construct T such that all road sections of G are contained in T. Since a road section is only incident to junction edges, this is always possible. In Section 5.5 we show that large parts of realistic road networks can actually be decomposed into paths and trees without losing optimality.

Milp. In order to provide upper bounds for the evaluation of our labeling algorithms, we implement a mixed-integer linear programming (MILP) model that solves MaxLabeledRoads optimally on arbitrary abstract road graphs. The basic idea is to discretize all possible label positions and to restrict the space of feasible solutions to non-overlapping sets of labels.

We now describe the MILP formulation in detail. To simplify the presentation, we drop the rather technical concept of *well-shaped* labels, but note that it can be easily incorporated into the MILP. In the following, let the edges of G be (arbitrarily) directed.

We first discretize the problem as follows. Two labels ℓ and ℓ' are *equivalent* if they cover the same edges in the same order, i.e., $\mathcal{P}_\ell = \mathcal{P}_{\ell'}$, and only their end points differ; see Figure 5.4(b). For each such equivalence class we create one label ℓ; we denote its equivalence class by $\mathcal{K}_\ell$. Further, let L denote the set of such created labels. The main idea of the MILP is to select a subset of L and to determine the exact positions of the labels' end points on their terminals such that they do not overlap and label a maximum number of road sections.

Now, consider a label $\ell \in L$ and the path $\mathcal{P}_\ell = (e_1, e_2, \cdots, e_{k-1}, e_k)$ that is covered by ℓ; see Figure 5.4(b). In the following, we call e_1 and e_k the *terminals* of ℓ and the others *internal edges* of ℓ. Assume that the head of the label ℓ lies on e_1 and the tail on e_k, then ℓ can slide along $\mathcal{P}_\ell$ changing the covered road sections until the head or tail of ℓ *hits* an end point of e_1 or e_k, respectively. At each position, ℓ coincides with an equivalent label ℓ'. Obviously, those labels exactly form $\mathcal{K}_\ell$. Further, there exist two positions on e_1 such that the head of ℓ has either minimum geodesic distance a or maximum geodesic distance b to the source of e_1, respectively. We define the interval $H_\ell = [a, b]$. Analogously, we define the interval T_ℓ for the tail of ℓ and the edge e_k.

For each label $\ell \in L$ we introduce the variables $x_\ell \in \{0, 1\}$, $h_\ell \in H_\ell$ and $t_\ell \in T_\ell$, and for each road section $e \in E$ the variable $y_e \in \{0, 1\}$. We interpret $x_\ell = 1$ such that ℓ is selected for the labeling. The variables h_ℓ and t_ℓ are interpreted as the geodesic distances of the head and tail to the source of the head's and tail's terminal, respectively; see Figure 5.4(c). We interpret $y_e = 1$ as road section e being labeled and maximize the sum $\sum_{e \in E} y_e$ subject to the following constraints.

For each $\ell \in L$ we require

$$\operatorname{cov}(e_1, \ell) + \operatorname{length}(e_2) + \ldots + \operatorname{length}(e_{k-1}) + \operatorname{cov}(e_k, \ell) = \operatorname{length}(\ell), \tag{5.1}$$

where $\mathcal{P}_\ell = (e_1, \ldots, e_k)$, $\operatorname{length}(\ell)$ denotes the given length of ℓ and $\operatorname{cov}(e, \ell)$ is a linear expression describing what length of e is covered by ℓ. This expression depends on which end point of e is covered, whether the head or tail of ℓ lies on e, and on the position variables h_ℓ and t_ℓ, respectively; we omit the technical definition. Further, for each pair $\ell', \ell \in L$ we require

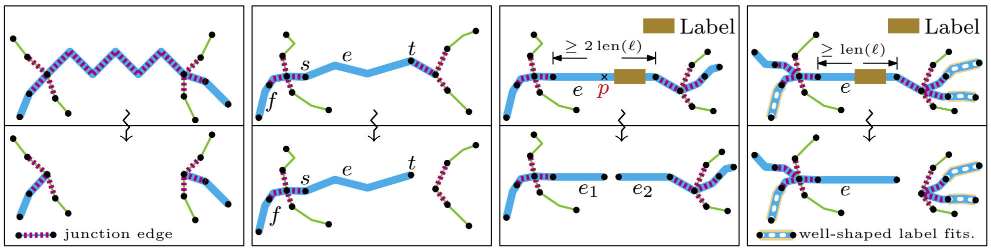
$$x_\ell + x_{\ell'} \leq 1 \quad \text{if an edge of } \ell \text{ is an internal edge of } \ell' \tag{5.2}$$

$$h_\ell - h_{\ell'} \leq M(2 - x_\ell - x_{\ell'}) \quad \text{if the heads of } \ell \text{ and } \ell' \text{ lie on a common terminal } e \text{ and } \ell \text{ covers the source of } e. \tag{5.3}$$

For each road section $e \in E$ and all labels $\ell_1, \ldots, \ell_k \in L$ labeling e we require

$$y_e \leq x_{\ell_1} + \cdots + x_{\ell_k} \tag{5.4}$$

Constraint (5.1) ensures that each label has the desired length $\operatorname{length}(\ell)$. Constraint (5.2) ensures that a label does not overlap another label internally, i.e., it (partly) covers an



(a) Rule 1. **(b)** Rule 2. **(c)** Rule 3. **(d)** Rule 4.

Figure 5.5: Illustration of the algorithms. Edges of the same color belong to the same road.

edge that is completely covered by another label. Constraint (5.3) ensures that labels ending on the same road section do not overlap on that edge, but ℓ ends on e before ℓ' starts; see Figure 5.4(c). Similar constraints are introduced for the other combinations on how heads and tails of ℓ and ℓ' can lie on a common edge, and on whether source or target of e is covered. For an appropriate large M the constraint is trivially satisfied if ℓ or ℓ' is not selected for the labeling. Finally, Constraint (5.4) ensures that road section e is only counted as labeled, if there is at least one selected label covering e.

Since L models all possible label positions and the constraints restrict the space of feasible solutions to non-overlapping sets of labels, it is clear that any optimal solution of the above MILP corresponds to an optimal solution of MaxLabeledRoads.

Theorem 5.1. *Milp solves MaxLabeledRoads optimally.*

Finding an optimal solution for a MILP formulation is NP-hard in general and remains NP-hard for the stated formulation, because MaxLabeledRoads is NP-hard. However, it turns out that in practice we can apply specialized solvers to find optimal solutions for reasonably sized instances in acceptable times, see Section 5.5.

5.4.2 Decomposition of Road Networks.

We may speed up both our heuristic Tree and the exact approach Milp by decomposing the road graph into smaller, independent components to be labeled separately, i.e., components whose individual optimal solutions compose to a conflict-free optimal solution of the initial road graph. Such a decomposition allows us to compute solutions in parallel with either of the above methods and it further decreases the total combinatorial complexity. The decomposition rules guarantee that the labelings of the components can always be merged without creating any label overlaps. We name this technique D&C.

Step 1 – Decomposition. For many road sections, e.g., long sections, of real-world road networks labels can be easily placed preserving the optimal labeling. We iterate through the edges of G and cut or remove some of them if one of the following rules applies. As a result the graph decomposes into independent connected components; see Figure 5.5(a)–(d). Let e be the currently considered edge and let R be the road of e.

Rule 1. If e is a junction edge and it cannot be completely covered by a well-shaped label, i.e., e is not well-shaped, then remove e.

Rule 2. If e is a road section that ends at a junction that is not connected to any other road section of R, then detach e from that junction.

Rule 3. If e is a road section, a well-shaped label ℓ fits on e, and e is at least twice as long as ℓ, then cut e at its midpoint.

Rule 4. If e is a road section, a well-shaped label ℓ fits on e, and e is connected to a junction that is only connected to road sections of R that may completely contain a well-shaped label, then detach e from that junction.

On each edge we apply at most one rule. If we apply *Rule 3* or *Rule 4* on an edge e, we call e a *long-edge.* Afterwards, we determine all connected components of the remaining graph G', which are then independently labeled.

Step 2 – Label Placement. For the constructed components we compute solutions in parallel with either of the above methods.

Step 3 – Composition. Finally, we compose the labelings of the second step to one labeling. Due to the decomposition, no two labels of different components can overlap. If a long-edge e is not labeled, we place a label on it, which is possible by definition. We adapt the algorithms of Step 2 such that they do not count labeled road sections that were created by *Rule 3*, but we count the corresponding long-edge in this step.

Correctness. We now prove the correctness of the approach. To that end we first formalize the presented rules. We assume that the edges of G are (arbitrarily) directed.

Rule 1. If e is a junction edge and it cannot be completely covered by a well-shaped label, i.e., e is not well-shaped, then remove e.

Rule 2. Let $R_e(v)$ be the set of road sections that belong to the same road as e, and that are reachable from v in G when only traversing junction edges. If $e = (s,t)$ is a road section and $R_e(u) = \{e\}$ for an $u \in \{s,t\}$, then remove the junction edge that is incident to u.

Rule 3. If $e = (s,t)$ is a road section, a well-shaped label ℓ fits on e, and e is twice as long as ℓ, then replace e by the road sections $e_1 = (s,u_1)$ and $e_1 = (u_2,t)$, where u_1 and u_2 are two new vertices at the midpoint p of e, e_1 is a sub-polyline of e from s to u_i and e_2 is a sub-polyline of e from u_2 to t. We mark e_1 and e_2 as *stubs* and call e a *long-edge.*

Rule 4. If $e = (s,t)$ is a road section, a well-shaped label ℓ fits on e and for at least one end node $u \in \{s,t\}$ the road sections in $R_e(u) \setminus \{e\}$ are all stubs, then remove the junction edge incident to u. We mark e as *stub* and call e a *long-edge.*

The next theorem shows that D&C combined with any optimal algorithm yields an optimal labeling.

Theorem 5.2. *Let G be an abstract road graph and let $\mathcal{L}$ be the resulting labeling after applying D&C combined with an algorithm that yields optimal labelings. An optimal labeling $\mathcal{L}'$ of G and $\mathcal{L}$ label the same number of road sections.*

Proof. Let $G = (V, E)$ be an abstract road graph and let $\mathcal{L}'$ be an optimal labeling of G, i.e., no more road sections can be labeled. We show that we can transform $\mathcal{L}'$ into a labeling $\mathcal{L}$ that is found by D&C, and, furthermore, $\mathcal{L}$ and $\mathcal{L}'$ label the same number of road sections. If not mentioned otherwise, we assume a label to be well-shaped.

Rule 1. Assume that we apply *Rule 1* on G by deleting a junction edge e that cannot be completely covered by a well-shaped label. By definition no label may end on a junction edge, but it must end on a road section. Thus, in any labeling the edge e cannot be covered by any label. We therefore can delete the edge preserving the optimal labeling, i.e., an optimal labeling of G and $G' = (V, E \setminus \{e\})$ label the same number of road sections.

Rule 2. Assume that we apply *Rule 2* on the edge e. Since e is the only edge in $R_e(u)$, the edge is the end of a road, i.e., all other edges incident to u cannot belong to the same road of e. Since e is a junction edge, no label may end on a junction edge, and labels may only cover edges of the same road, no label can cover e in any labeling. We therefore can delete the edge preserving the optimal labeling.

Rule 3. Assume that we apply *Rule 3* on the road section $e = (s, t)$ splitting e into the edges $e_1 = (s, u_1)$ and $e_2 = (u_2, t)$. Since e may contain a well-shaped label, the road section e must be labeled in $\mathcal{L}'$.

If e is only labeled by labels that are completely contained in e, i.e., they do not cover other edges of G, we will find one of those labels in the composition step of D&C.

Hence, assume that there is a label $\ell_1 \in \mathcal{L}'$ that covers e and s. Since e is twice as long as the label length of e, this label cannot cover the location of $u_1 (u_2)$. The same applies for a label $\ell_2 \in \mathcal{L}$ that covers e and t. Since e is labeled by ℓ_1 (ℓ_2) we can remove all other labels that only label e without changing the maximum number of labeled road sections. Hence, the point at u_1 is not covered by any label, which means we can split e at this point preserving the optimal labeling.

Rule 4. Assume that we apply *Rule 4* on the road section $e = (s, t)$ with $u = s$; same arguments hold for $u = t$. Hence, the road sections in $R_e(u) \setminus \{e\}$ are all stubs, i.e., well-shaped labels can placed on any of these road sections. Let j be the junction edge that is connected to s. Assume that there is a label ℓ that labels e and an edge e' of $R_e(u) \setminus \{e\}$ such that u is covered by ℓ.

If e and e' are also labeled by other labels, we can remove ℓ without changing the number of labeled road sections and remove j. So assume that e is not labeled by another label. In that case we remove ℓ and place a label that completely lies on e without covering any other edges; by definition of the rule this is possible. If e' is also not labeled by any other label, we also place a label on e', which is possible, because e' is a stub. Hence, we can remove j preserving the optimal labeling. □

5.5 Evaluation

We evaluate our framework and in particular the performance of our new tree-based labeling heuristic by conducting a set of experiments on the road networks of 11 North American and European cities; see Table 5.1. While the former ones are characterized by grid-shaped road networks, the latter ones rarely posses such regular geometric structures. Since the road networks in rural areas are much sparser than those of cities, we refrained from considering these networks and focused on the more complex city road networks. We extracted the abstract road graphs from the data provided by OpenStreetMap[1]. We applied the spherical Mercator projection ESPG:3857, which is also known as *Web Mercator* and used by several popular map-services. We considered the three scale factors 4.773, 2.387 and 1.193, which approximately correspond to the map scales 1:16000, 1:8000, 1:4000[2]. Further, they correspond to the *zoom levels* 15, 16 and 17, respectively, which are widely used by map services as OpenStreetMap. Those zoom levels show road networks in a size that already allows labeling single road sections, while the map is not yet so large that it becomes trivial to label the roads. We applied the standard drawing style for OpenStreetMap, which in particular includes the stroke width and color of roads as well as the font size of the labels. Further, this specifies for each zoom level the considered road categories; the higher the zoom level the more categories are taken into account.

Our implementation is written in C++ and compiled with GCC 4.8.4 using optimization level `-O3`. MILPs were solved by Gurobi[3] 6.0. The experiments were performed on a 4-core Intel Core i7-2600K CPU clocked at 3.4 GHz, with 32 GiB RAM. The D&C-approach labels single components in parallel. For computing the Delaunay triangulation we used the library Fade2d[4].

For each city and each zoom level we applied the algorithms BaseLine, Tree, D&C+Tree, Milp and D&C+Milp. We adapted the algorithm such that short road sections (shorter than the width of the letter `W`) are not counted, because they are rarely visible. Further, we let Mapnik (Version 3.0.9) render the same input. For each label we identified for each of its letters the closest road section r with the same name and counted it as labeled. Since Mapnik does not optimize the labeling by the same criteria as we do, we compensate this by also counting neighboring road sections as labeled if the junction in between them is not incident to any other road section. This accounts for those long road sections that we split artificially to resolve Issue 3.

The raw data of our experiments is made available on

`i11www.iti.kit.edu/roadlabeling`

[1]http://www.openstreetmap.org
[2]http://wiki.openstreetmap.org/wiki/Zoom_levels
[3]http://www.gurobi.com
[4]http://www.geom.at

Table 5.1: Statistics for Baltimore (BA), Berlin (BE), Boston (BO), Los Angeles (LA), London (LO), Montreal (MO), Paris (PA), Rome (RO), Seattle (SE), Vienna (VI) and Washington (WA) for zoom level 15, 16 and 17. *OSM*: Number of input segments in thousands. *Segments:* Percentage of segments after Phase 1, Step 3 in relation to input segments. *Graph*: Number of road sections after Phase 1 in thousands. *Time:* Running time for Phase 1.

		European Cities					North American Cities					
		BE	LO	PA	RO	VI	BA	BO	LA	MO	SE	WA
Zoom 15	OSM	143.9	437.6	225.1	87.7	85.1	196.1	174.5	257.1	134.6	315.3	82.2
	Segm.	62	80	65	66	63	52	54	74	78	70	39
	Graph	28.5	78.5	35.3	10.3	14.8	24.7	20.1	61.3	31.9	63.1	8.7
	Time	16	62	28	10	10	22	19	42	20	40	8
Zoom 16	OSM	225.0	563.4	292.5	117.0	119.9	332.1	225.0	327.0	161.4	433.1	103.9
	Segm.	55	73	62	62	54	40	50	67	72	59	37
	Graph	37.9	105.4	49.9	15.4	18.9	33.8	27.8	80.6	40.2	77.1	11.4
	Time	21	65	32	12	11	28	21	44	21	42	9
Zoom 17	OSM	225.0	563.4	292.5	117.0	119.9	332.1	225.0	327.0	161.4	433.1	103.9
	Segm.	64	80	69	70	60	46	56	73	83	64	43
	Graph	47.1	127.0	59.1	19.4	22.3	39.5	32.3	90.4	47.4	87.9	13.0
	Time	24	67	33	13	11	29	22	46	22	43	10

On this page we also provide interactive maps of the cities Berlin, London, Los Angeles and Washington, which present the computed labelings.

Phase 1. With a maximum of 67 seconds (London, zoom 17) and 27 seconds averaged over all instances, Phase 1 can be applied on large instances in reasonable time. During Phase 1 the number of segments is reduced to between 40% and 83% of the original instance (measured after Step 3, before creating junction edges); see Table 5.1. This clearly indicates that the procedure aggregates many lanes, since by design the approach does not change the overall geometry, but the simplification maintains the shape of the original network. This is also confirmed by the labelings; see Figure 5.1(b)–(c) and interactive maps.

Phase 2, Running Time. We first consider the average running times over all zoom levels; see Figure 5.6(a). We did not measure the running times of Mapnik, because its labeling procedure is strongly interwoven with the remaining rendering procedure, which prevents a fair comparison. As to be expected MILP is the slowest method (max. 126 sec., Los Angeles, ZL 15), while BASELINE is the fastest procedure (max. 0.17 sec.). Combining MILP with D&C results in an average speedup of 2.29 over all instances and a maximum speedup of 3.44; see Table 5.2.

The algorithm TREE needs less than 4.7 seconds and its median is about 1.3 seconds. Hence, despite its worst-case cubic asymptotic running time, it is fast in practice. Similar to MILP, it is further enhanced by combining it with D&C for a speedup of

Table 5.2: *Speedup*: Ratio of running times of two algorithms. *Quality:* Ratio of the number of labeled road sections computed by two algorithms.

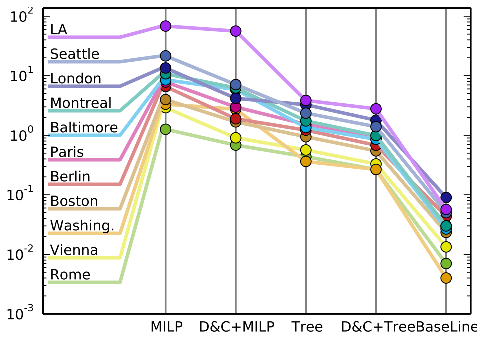
		European Cities					North American Cities						
	Ratio	BE	LO	PA	RO	VI	BA	BO	LA	MO	SE	WA	**Avg.**
Speedup	$\frac{\text{Milp}}{\text{D\&C+Milp}}$	3.44	3.07	2.51	1.71	3.12	1.44	2.33	1.3	1.79	3.1	1.32	**2.29**
	$\frac{\text{Tree}}{\text{D\&C+Tree}}$	1.77	1.8	1.73	1.62	1.71	1.57	1.71	1.37	1.75	1.68	1.35	**1.64**
	$\frac{\text{D\&C+Milp}}{\text{D\&C+Tree}}$	2.82	2.32	3.33	2.54	2.74	6.84	3.06	21.59	6.36	5.32	10.59	**6.14**
Quality	$\frac{\text{D\&C+Tree}}{\text{Tree}}$	1.01	1.0	1.0	1.0	1.01	1.01	1.0	1.01	1.02	1.01	1.02	**1.01**
	$\frac{\text{D\&C+Tree}}{\text{Milp}}$	1.0	1.0	0.99	0.99	0.99	0.96	0.99	0.96	0.97	0.97	0.91	**0.97**
	$\frac{\text{Mapnik}}{\text{Milp}}$	0.74	0.85	0.83	0.91	0.76	0.71	0.8	0.62	0.61	0.8	0.68	**0.75**
	$\frac{\text{BaseLine}}{\text{Milp}}$	0.58	0.49	0.4	0.38	0.48	0.39	0.42	0.39	0.46	0.37	0.24	**0.42**
	$\frac{\text{D\&C+Tree}}{\text{Mapnik}}$	1.36	1.19	1.2	1.09	1.29	1.37	1.25	1.55	1.58	1.21	1.33	**1.31**

1.64 with respect to Tree, and an average speedup of 6.14 with respect to D&C+Milp; see Table 5.2. In the latter case it even has a maximum speedup of about 21.6. Since decomposing and composing the labelings is done sequentially, the theoretically possible speed up using D&C is not achieved.

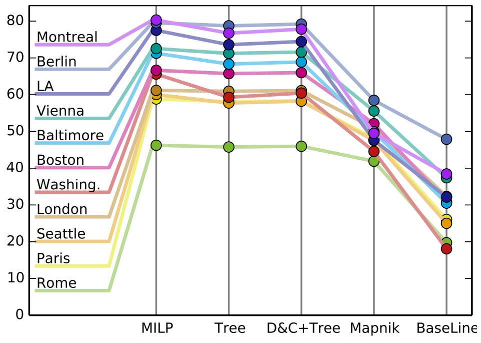
If we break down the running times into single zoom levels, we observe similar results; see e.g., Figure 5.7. Since with increasing zoom level the instance size grows, for most of the algorithms also the running time increases. Only for North American cities and Milp we observe that the running time for instances of smaller zoom levels are higher than for larger zoom levels.

Phase 2, Quality. First we analyze the average percentage of labeled road sections over the three zoom levels; see Figure 5.6(b). As an upper bound, Milp, which provably solves MaxLabeledRoads optimally, yields results from 46.2% (Rome) to 80.3% (Montreal). Considering zoom levels independently, we obtain a minimum of 27.5% (Rome, ZL 15) and a maximum of 91.7% (Montreal, ZL 17). We think that the wide span is attributed to the different structures of road networks and road names, e.g., Rome has a lot of short alleys and long road names. Hence, many road components are too short or convoluted to contain a single label. Abbreviating road names could help to overcome this problem.

The algorithm D&C+Tree yields marginally better results than Tree, but only 1% on average, see Table 5.2. Comparing D&C+Tree with Milp we observe that D&C+Tree yields near-optimal results with respect to our road-section based model. On average it reaches 97% of the optimal solution; see Table 5.2. While the quality ratio is only 91% for Washington, more than half of the instances are labeled with a quality ratio of ≥ 99%. For European cities the percentage of road sections that belong



(a) Running time (seconds).



(b) Percentage of labeled road sections.

Figure 5.6: (a) Running times in seconds of the algorithms (logarithmic scale). (b) Percentage of labeled road sections over all zoom levels broken into the different algorithms.

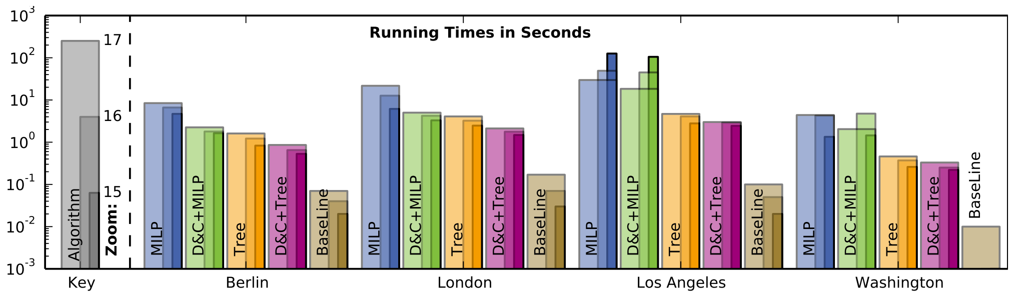


Figure 5.7: Running times of the algorithms broken down in zoom levels and algorithms. The width of the bars (thin, mediate, wide) corresponds with zoom levels (15, 16, 17).

to components that are optimally solved by Tree (long edges, paths, and trees) is notably higher than those for North American cities; see Figure 5.8. Nonetheless, we obtain similar percentages of labeled road sections for North American Cities. Hence, the heuristic computing a spanning tree of non-tree components is both fast and yields near-optimal results. The additional implementation effort of Tree is further justified by the observation that the naive way to place labels only on single road sections lags far behind; only 42% on average, 58% as maximum and 24% as minimum compared to the optimal solution. Mapnik achieves on average 75% of the optimal solution and a maximum of 91%. For more than the half of the instances Mapnik achieves at most 76% of the optimal solution. So in direct comparison, D&C+Tree consistently outperforms Mapnik. Moreover, D&C+Tree has a better utilization of labels and achieves an average ratio of 1.61 labeled road sections per label, compared to Mapnik with a ratio of 1.37; see Figure 5.9.

With increasing zoom level the number of labeled road sections is increased, which is to be expected, since more road sections become long-edges; see Figure 5.10 for four

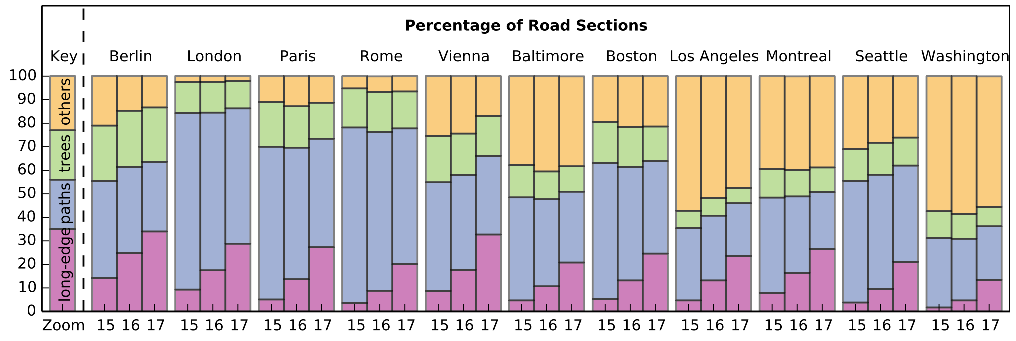
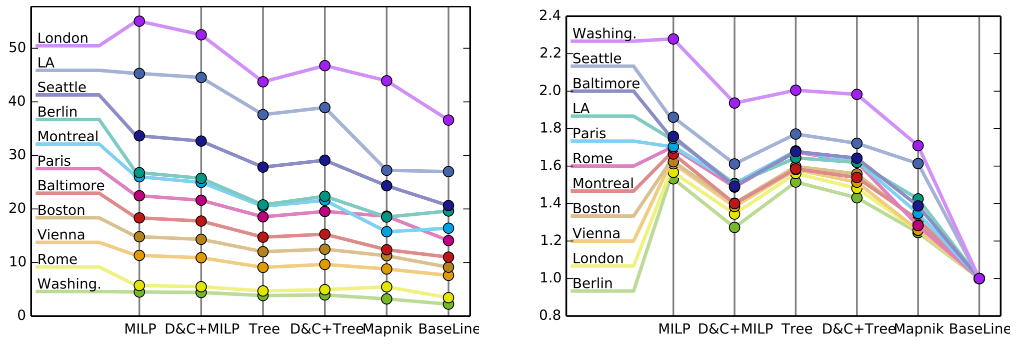


Figure 5.8: Decomposition of road networks by D&C. Percentage of long-edges, road sections in paths, trees and other components in which the road networks is decomposed.



(a) **(b)**

Figure 5.9: (a) Number of placed labels in thousands. (b) Ratio of labeled road sections and placed labels: (#Labeled Road Sections)/(#Labels).

cities (similar results apply for the others) and Figure 5.8. For each zoom level, we observe similar results as described before: Tree and D&C+Tree achieve near-optimal solutions and Mapnik labels considerably fewer road sections. However, for smaller zoom levels the gap between Milp and Mapnik shrinks.

From a visual perspective, labels lie on the skeleton of the road network, which is achieved by design; see Figure 5.1(c) and the interactive maps. Instead of unnecessary repetition of labels, labels are only placed if they actually convey additional information. In particular, visual components are labeled, but not single lanes that are indistinguishable due to the zoom level.

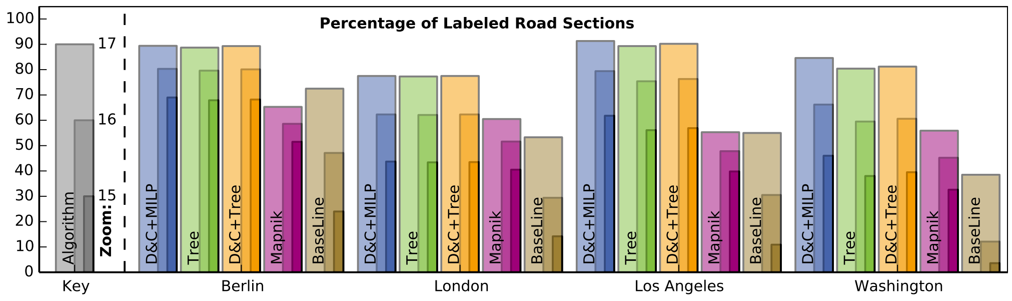


Figure 5.10: Percentage of labeled road sections broken down in zoom levels and algorithms. The width of the bars (thin, medium, wide) corresponds to the zoom level (15, 16, 17).

5.6 Conclusions

We introduced a framework for labeling road maps based on an abstract road graph model that is combinatorial rather than geometric. We showed in our experimental evaluation that our proposed heuristic for decomposing the road graph into tree-shaped subgraphs and labeling those trees provably optimally is efficient and effective. It has running times in the range of seconds to one minute even for large road networks such as London with more than 100,000 road sections. Moreover, it achieves near-optimal quality ratios (on average 97%) compared to upper bounds computed by the exact method Milp. Our algorithm clearly outperforms the labeling algorithm of the standard OSM rendering engine Mapnik, with an average improvement in the number of labeled road sections by 31%. Interestingly, Milp is able to compute mathematically optimal solutions within a few minutes for all our test instances, even though it is slower by a factor of about 6 compared to the tree-based algorithm. For practical purposes there is a trade-off between a final, but rather small improvement in quality at the cost of a significant and by the very nature of Milp unpredictable increase in running time. We only implemented essential cartographic criteria to evaluate the algorithmic core of our framework; further criteria (e.g., abbreviated names) and alternative definitions of road sections can be easily incorporated. The framework can further be pipelined with labeling algorithms for other map features, e.g., after placing labels for point features, one may block all parts of the road network covered by a point label and label the remaining road network such that no labels overlap. While this allows the labeling of different types of features sequentially, constructing a labeling of all features in one single step remains an open problem.

6 Temporal Map Labeling: Model and Theory

Abstract. In this chapter, we consider map labeling for the case that a map undergoes a sequence of operations such as rotation, zoom and translation over a specified time span. We unify and generalize several previous models for dynamic map labeling into one versatile and flexible model. In contrast to previous research, we completely abstract from the particular operations and express the labeling problem as a set of time intervals representing the labels' presences, activities and conflicts. One of the model's strength is manifested in its simplicity and broad range of applications. In particular, it supports label selection both for map features with fixed position as well as for moving entities (e.g., for tracking vehicles in logistics or air traffic control).

We study the active range maximization problem in this model. We prove that the problem is NP-complete and W[1]-hard, and present constant-factor approximation algorithms. In the restricted, yet practically relevant case that no more than k labels can be active at any time, we give polynomial-time algorithms as well as constant-factor approximation algorithms. In the subsequent chapter we experimentally evaluate our approach.

This chapter is based on and partly taken from joint work with Lukas Barth, Andreas Gemsa, Martin Nöllenburg and Darren Strash [GNN13b, GNN13a, Bar+16].

6.1 Introduction

Dynamic digital maps are becoming more and more ubiquitous, especially with the rising numbers of location-based services and smartphone users worldwide. Consumer applications that include personalized and interactive map views range from classic navigation systems to map-based search engines and social networking services. Likewise, interactive digital maps are a core component of professional geographic information systems. All these map services have in common that the content of the map view is changing over time based on interaction with the system (i.e., zooming, panning, rotating, content filtering, etc.) or the physical movement of the user or a set of tracked entities.

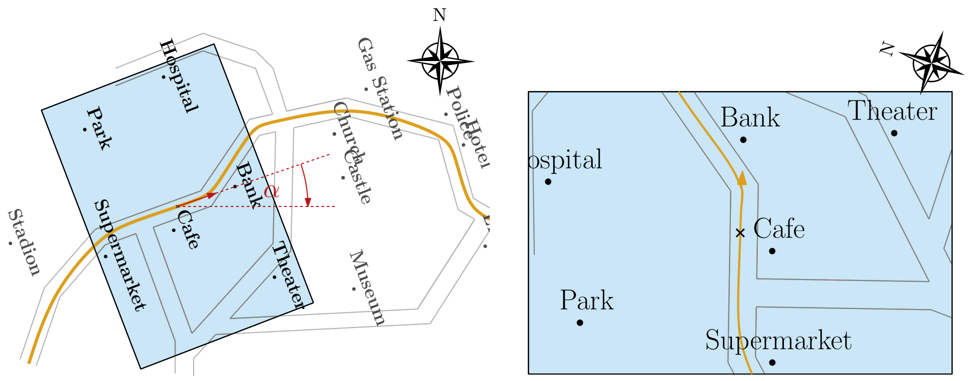
In the past decade, dynamic map labeling has also captured the interest of researchers, leading to the study of labeling problems in maps that support certain subsets of operations like zooming, panning, and rotation. The main difficulty in dynamic maps is that the selection and placement of labels must be temporally coherent (or *consistent*) during all map animations resulting from interactions, rather than being optimized individually for each map view as in static map labeling. A map with temporally coherent labeling avoids visually distracting effects like jumping or flickering labels [BDY06]. Consistent dynamic map labeling problems are typically NP-hard and approximation results as well as heuristics are known [BDY06, Bee+10, GNR16a, GNR16b, LLP14]. However, most existing algorithmic results in dynamic map labeling take a *global* view on the map, which optimizes over the *whole* interaction space, regardless of which portion of that space is actually explored by the user. For a detailed introduction to dynamic map labeling and related research see Chapter 3.

Contribution and Outline. In this chapter, we take a more *local* view on dynamic map labeling. Our aim is to develop algorithms that optimize the labeling for a specific map animation given offline as an input. Any feature or label that is not relevant for that particular animation—for example, because it never enters the map view—can be ignored by our algorithms. This approach not only allows us to compute better labelings by removing unnecessary dependencies and non-local effects, but it also reduces the problem size, since fewer features and labels must be taken into account.

In Section 6.2, we first formulate an abstract, generic model for offline, temporal labeling problems, in which labels and potential conflicts between labels are represented as intervals over time. To represent a label's presence, we use a *presence* interval, which corresponds to the time that a label is present (but not necessarily displayed) in the map view. That is, whenever a label enters the map view, a corresponding presence interval starts, and whenever a label leaves the view, its current presence interval ends. Next, a *conflict interval* (or simply *conflict*) between two present labels starts and ends at the points in time at which the two labels start and stop intersecting. A temporal labeling is then simply represented as a set of subintervals—the labels' *activity intervals*, during which the labels are displayed, where no two conflicting labels are displayed simultaneously. Depending on the objective and additional consistency constraints of the labeling model, different sets of subintervals may be chosen by the algorithm.

This is a very versatile model, which includes, for instance, map labeling for car navigation systems, in which the map view changes position, angle, and scale according to the car's position, heading, and speed following a particular route; see Figure 6.1. To give another, seemingly different example, it also includes the problem of labeling a set of moving entities in a map view (e.g., for tracking vehicles in logistics or planes in air traffic control). Also non-map related applications such as labeling 3D scenes as they occur in medical information systems is covered by our model. Put differently, the model comprises any application in which start and end times of label presences and conflicts can be determined in advance. Further, the conflicts are not restricted to label-label conflicts but may also include label-object conflicts. See Chapter 7 for a more detailed discussion of applications.

Based on this model we introduce the two optimization problems GeneralMax-

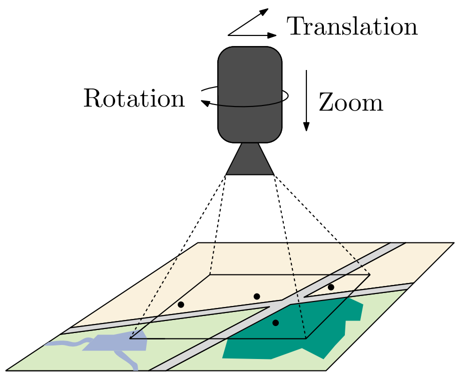


(a) Overall view. **(b)** Viewport.

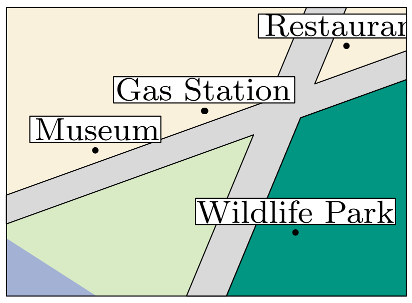
Figure 6.1: Trajectory-based labeling. (a) Viewport moves and aligns along a given trajectory (fat orange line). Labels align to the viewport. (b) The user's view on the scene.

Total and k-RestrictedMaxTotal. While in the former one we simply maximize the number of visible labels integrated over time, in the latter one we further require that at most k labels are active at any time for some constant k. We note that limiting the number of simultaneously active labels is of practical interest as to avoid overly dense labelings, in particular for dynamic maps on small-screen devices such as in car navigation systems. We further refine our model by the three activity models AM1, AM2 and AM3, which introduce special requirements to enforce temporal consistency avoiding flickering when switching labels on and off.

In Section 6.3, we investigate the problem GeneralMaxTotal. We first prove that GeneralMaxTotal is NP-complete; in fact it is even W[1]-hard and thus it is unlikely that a fixed-parameter tractable algorithm exists. For the special case of unit-square labels, we give an efficient approximation algorithm with different approximation ratios depending on the actual label activity model. In Section 6.4 we present polynomial-time algorithms for k-RestrictedMaxTotal in AM1 and AM2. For AM3 we show that the problem is NP-hard for $k \geq 2$. Further, for k-RestrictedMaxTotal we present efficient constant-factor approximation algorithms for all three activity models assuming that all labels are unit-squares. In Chapter 7, we experimentally evaluate the presented model.



(a) Overall view.



(b) Viewport.

Figure 6.2: Model. (a) Overall view of the scene. Depending on the rotation, translation, and zoom, the camera shoots a restricted part of the scene. The objects to be labeled are represented by black dots. (b) The corresponding viewport of the camera. The labels are placed near their objects.

6.2 Model

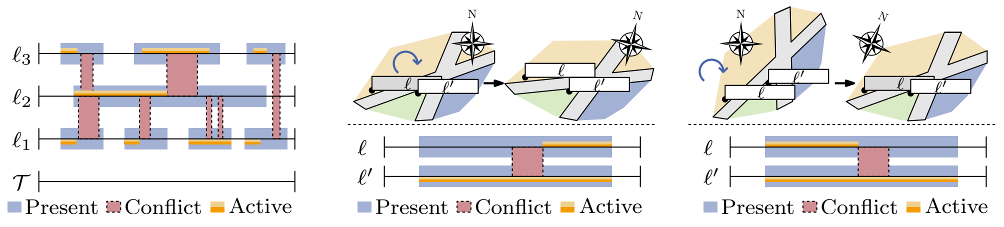
We now formally describe the temporal labeling model that we use in the remainder of this chapter. It particularly unifies and generalizes the models presented by Been et al. [BDY06], and de Berg and Gerrits [BG13].

6.2.1 Basic Model

We are given a set $O = \{o_1, \ldots, o_n\}$ of objects in a scene over a given time span $\mathcal{T} = [0, 1]$. Further, for each object o we are given a label ℓ, e.g., text describing o. We denote the set of labels by $L = \{\ell_1, \ldots, \ell_n\}$, where ℓ_i is the label of o_i. To quantify the importance of a label, we define for each label $\ell \in L$ a positive weight $w_\ell \in \mathbb{R}^+$.

We have a restricted view on the scene through a camera, i.e., the objects are projected onto an infinite plane P such that we can only see a restricted section V of P, where V models the *viewport* of the camera; see Figure 6.2 for an example. During the time interval $\mathcal{T}$, the objects are moving and the camera changes its perspective by changing its position, direction and zoom. We denote the plane P and the viewport V at time t by $P(t)$ and $V(t)$, respectively. Depending on the position of the object $o_i \in O$, each label ℓ_i has a certain shape and position on $P(t)$ at time t; we denote the geometric shape of ℓ at time t by $\ell(t)$. Following typical map labeling models we may assume that $\ell(t)$ is a (closed) rectangle enclosing the text; one may also assume other shapes. In the following, we introduce some further notations to describe the setting precisely.

According to the perspective and position of the camera, not every label $\ell(t)$ is



(a) Valid activity. **(b)** End of conflict. **(c)** Start of conflict.

Figure 6.3: Label activity. The intervals illustrate presence, conflict and activity intervals. (a) A set of valid activity intervals. (b)–(c) The maps rotate clockwise, while the labels keep aligned horizontally. White labels are active, while gray labels are inactive. The witness label ℓ' justifies (b) the start (c) the end of ℓ's activity interval.

contained in the viewport at time t. We say that a label ℓ is *present* at time t if $\ell(t)$ is (partly) contained in $V(t)$; that is, $\ell(t) \cap V(t) \neq \emptyset$. We assume that the time intervals, during which a label ℓ is present, are given by a set Ψ_ℓ of disjoint, closed sub-intervals of $\mathcal{T}$; see Figure 6.3. For such an interval $[a,b] \in \Psi_\ell$ we also write $[a,b]_\ell$ indicating that it belongs to ℓ. We denote the union of all those sets Ψ_ℓ by Ψ and assume that Ψ is a multi-set, as it may contain the same interval $[a,b]$ multiple times, where each occurrence of $[a,b]$ belongs to a different label.

Two labels ℓ and ℓ' are in *conflict* at time $t \in \mathcal{T}$, if the geometric shapes of both labels intersect, i.e., $\ell(t) \cap \ell'(t) \neq \emptyset$. We describe the occurrences of conflicts between two labels $\ell, \ell' \in L$ by a set of closed intervals, $C_{\ell,\ell'} = \{[a,b] \subseteq \mathcal{T} \mid [a,b]$ is maximal and ℓ and ℓ' are in conflict at all $t \in [a,b]\}$. For such an interval $[a,b] \in C_{\ell,\ell'}$ we also write $[a,b]_{\ell,\ell'}$ indicating that it is a *conflict interval* between ℓ and ℓ'. We denote the set of all conflict intervals over all pairs of labels by the multi-set C.

To avoid overlaps between labels, we display a label ℓ only at certain times when no other displayed label overlaps ℓ; the label ℓ is said to be *active* at those times. We describe the activity of ℓ, by a set Φ_ℓ of disjoint intervals[1]. For such an interval $[a,b] \in \Phi_\ell$ we also write $[a,b]_\ell$ to indicate that the *activity interval* belongs to ℓ. The union of all activity intervals over all labels is denoted by the multi-set Φ.

We say that two activity (presence) intervals $[a,b]_\ell$ and $[c,d]_{\ell'}$ of two labels ℓ and ℓ' are in *conflict* if there is a time t in the intersection of the open intervals $(a,b) \cap (c,d)$ such that the labels ℓ and ℓ' are in conflict at t.

An instance of temporal labeling is then defined by the set L of labels, the set Ψ of presence intervals and the set C of conflict intervals. We thus completely abstract away the geometry of the problem, while all essential information of the temporal

[1]Technically, one needs to distinguish between open and closed intervals, i.e., for closed rectangular labels, the presence and conflict intervals are closed but the activity intervals are open. However, including or excluding the interval boundaries makes no difference in our algorithms and hence we decided to simply use the notation $[a,b]$ for all respective intervals unless stated otherwise.

labeling instance is captured combinatorially in Ψ and C. In this chapter, we primarily focus on conflict-free label selection, and therefore assume that Ψ and C are given as input. In Chapter 7 we describe how to construct Ψ and C for the specific application of navigation systems.

Similarly to Been et al. [BDY06] for a temporal labeling we require the following temporal consistency criteria:

(C1) A label should not be set active and inactive repeatedly to avoid *flickering*.

(C2) The position and size of a label should be changed continuously, it should not *jump*.

(C3) Labels should not overlap.

We formalize these consistency criteria and say the activity set Φ is *valid* (see Figure 6.3(a)) if

(R1) for each activity interval $I_\ell \in \Phi$ there is a presence interval $I'_\ell \in \Psi$ with $I_\ell \subseteq I'_\ell$,

(R2) for each presence interval $I_\ell \in \Psi$ there is at most one activity interval $I'_\ell \in \Phi$ with $I'_\ell \subseteq I_\ell$, and

(R3) no two activity intervals of Φ are in conflict.

Requirement (R1) enforces that a label is only displayed if it is present in the viewport. Requirement (R2) prevents a label from *flickering* during a presence interval (C1), while (R3) enforces that no two displayed labels overlap (C3). In fact, (R2) is only a minimum requirement for avoiding flickering labels, which we later extend to stronger variants. By assuming that labels' positions are fixed relative to their anchors, labels may not *jump* (C2) as long as labeled objects are either fixed or move continuously. From now on we assume that an activity set is valid, unless we state otherwise.

6.2.2 Optimization Problems

Based on the introduced model we investigate two optimization problems for temporal labeling that aim to maximize the overall active time of labels. The first problem allows for any number of labels to be active at the same time, and the second allows at most k labels to be active at the same time, which reduces the amount of presented information. We define the weight of an activity interval $[a,b]_\ell \in \Phi$ to be $w([a,b]_\ell) = (b-a) \cdot w_\ell$.

Problem 6.1 (GeneralMaxTotal).
Given: *Instance* (L, Ψ, C).
Find: *Activity set* Φ *maximizing* $\sum_{[a,b]_\ell \in \Phi} w([a,b]_\ell)$.

Figure 7.8(a) in Chapter 7 shows an example of a single frame of a temporal labeling that is optimal with respect to GeneralMaxTotal. While such a labeling is acceptable for general applications such as spatial data exploration, for small-screen devices, such as car navigation systems, the same labeling may overwhelm or distract the user with too much additional information. In fact, psychological studies have shown that untrained users are strongly limited in receiving, processing, and remembering

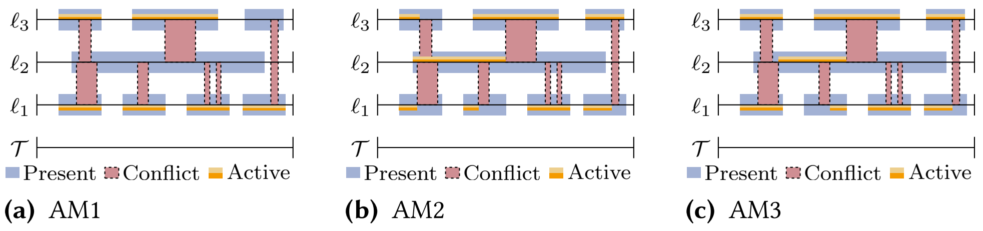


Figure 6.4: The activity models AM1, AM2 and AM3.

information (e.g., see [Mil56]). For applications that do not receive a user's full attention it is therefore desirable to restrict the number of simultaneously displayed labels, which we formalize as an alternative optimization problem as follows.

Problem 6.2 (***k***-RestrictedMaxTotal).
Given: *Instance* (L, Ψ, C), $k \in \mathbb{N}$.
Find: *Activity set* Φ *maximizing* $\sum_{[a,b]_\ell \in \Phi} w([a,b]_\ell)$, *s.t. at any time* t *at most* k *labels are active.*

6.2.3 Activity Models

So far labels may become active or inactive within the viewport without any external influence, see, e.g., the second activity interval of ℓ_3 in Figure 6.3(a). Hence, the activity behavior of labels, even in an optimal solution Φ, is not necessarily explainable to a user by simple and direct observations such as "the label becomes inactive at time t, because at that moment an overlap starts with another active label". The absence of those simple logical explanations may lead to unnecessary irritations of the user. To account for that we introduce the concept of justified activity intervals.

Consider a label ℓ with activity interval $[a, b]_\ell \in \Phi$. We say that the start of $[a, b]_\ell$ is *justified* if ℓ enters the viewport at time a or if there is a *witness* label ℓ' such that a conflict of ℓ and ℓ' ends at a and ℓ' is active at a; see Figure 6.3(b).

Analogously, we say that the end of $[a, b]_\ell$ is *justified* if ℓ leaves the viewport at time b or if there is a witness label ℓ' such that a conflict of ℓ and ℓ' begins at b and ℓ' is active at b; see Figure 6.3(c). If both the start and end of $[a, b]_\ell$ are justified, then $[a, b]_\ell$ is justified.

We distinguish the three activity models AM1, AM2, and AM3 that consider justified activity intervals; see Figure 6.4. While for AM1, a label may only become active and inactive when it enters and leaves the viewport, for AM2 it may also become inactive before leaving the viewport if a witness label justifies this event. AM3 further allows a label to become active after entering the viewport if a witness label justifies that event.

AM1. An activity Φ satisfies AM1 if any activity interval $[a, b]_\ell \in \Phi$ is justified and there is a presence interval $[c, d]_\ell \in \Psi$ of the same label ℓ with $[a, b]_\ell = [c, d]_\ell$.

AM2. An activity Φ satisfies AM2 if any activity interval $[a, b]_\ell \in \Phi$ is justified and there is a presence interval $[c, d]_\ell \in \Psi$ of the same label ℓ with $a = c$.

AM3. An activity Φ satisfies AM3 if any activity interval $[a, b]_\ell \in \Phi$ is justified.

We have described only the core of the model. Depending on the application it can be easily extended to more complex variants, e.g., requiring minimum activity times.

6.3 Solving GeneralMaxTotal

In this section we discuss GeneralMaxTotal. First, we present results on its computational complexity and then we present a simple approximation algorithm for the case that the labels are unit squares.

6.3.1 Computational Complexity

We first prove that the according decision problem of GeneralMaxTotal is NP-complete with respect to the three activity models. The membership in NP follows from the fact that the start and the end of an active interval must coincide with the start or end of a presence interval or a conflict interval. Thus, there is a finite number of candidates for the endpoints of the active intervals so that a solution $\mathcal{L}$ can be guessed. Verifying that $\mathcal{L}$ is valid in one of the three models and that its value exceeds a given threshold can obviously be checked in polynomial time.

For the NP-hardness we apply a straight-forward reduction from the NP-complete maximum independent set of rectangles problem [FPT81]. We simply interpret the set of rectangles as a set of labels with unit weight, choose a short vertical trajectory T and a viewport R that contains all labels at any point of T. Since the conflicts do no change over time, the reduction can be used for all three activity models. By means of the same reduction and Marx' result [Mar05] that finding an independent set for a given set of axis-parallel unit squares is W[1]-hard we derive the next theorem.

Theorem 6.1. *GeneralMaxTotal is NP-hard and W[1]-hard for all activity models AM1–AM3. In particular, the respective decision problem is NP-complete for all activity models AM1–AM3.*

As a consequence, GeneralMaxTotal is not fixed-parameter tractable unless W[1]=FPT. Note that this also means that for k-RestrictedMaxTotal we cannot expect to find an algorithm that runs in $O(p(n) \cdot C(k))$ time, where $p(n)$ is a polynomial that depends only on the number n of presence and conflict intervals, and the computable function $C(k)$ depends only on the parameter k.

6.3.2 Approximation of GeneralMaxTotal

We now describe a simple greedy algorithm for GeneralMaxTotal in all three activity models assuming that all labels are unit squares anchored at their lower-left corner. Further, we assume that the weight of each presence interval $[a, b]_\ell$ is its length $w([a, b]_\ell) = b - a$.

Starting with an empty solution Φ, our algorithm GreedyMaxTotal removes the longest interval I from Ψ and adds it to Φ, i.e., the label of I is set active during I. Then, depending on the activity model, it updates all presence intervals that have a conflict with I in Ψ and continues until the set Ψ is empty.

For AM1 the update process simply removes all presence intervals from Ψ that are in conflict with the newly selected interval I. For AM2 and AM3 let $I_j \in \Psi$ and let $I_j^1, \ldots, I_j^k$ be the longest disjoint sub-intervals of I_j that are not in conflict with the selected interval I. We assume that $I_j^1, \ldots, I_j^k$ are sorted by their left endpoint. The update operation for AM2 replaces every interval $I_j \in \Psi$ that is in conflict with I with I_j^1. In AM3 we replace I_j by I_j^1, if I_j^1 is not fully contained in I. Otherwise, I_j is replaced by I_j^k. Note that this discards some candidate intervals, but the chosen replacement of I_j is enough to prove the approximation factor. Note that after each update all intervals in Ψ are valid choices according to the specific model. Hence, we can conclude that the result Φ of GreedyMaxTotal is also valid in that model.

In the following, we analyze the approximation quality of GreedyMaxTotal. To that end we first introduce a purely geometric packing lemma. Similar packing lemmas have been introduced before, but to the best of our knowledge for none of them it is sufficient that only one prescribed corner of the packed objects lies within the container.

Lemma 6.1. *Let C be a circle of radius $\sqrt{2}$ in the plane and let Q be a set of non-intersecting closed and axis-parallel unit squares with their bottom-left corner in C. Then Q cannot contain more than eight squares.*

Proof. First, we show that Q cannot contain more than nine squares and extend the result to the claim of the lemma. We begin by proving the following claim.

(S) At most three squares of Q can be stabbed by a vertical line. In order to prove (S) let $Q' \subseteq Q$ be a set of squares that is stabbed by an arbitrary vertical line l and let q_t be the topmost square stabbed by l and let q_b be the bottommost square stabbed by l. Since both the bottom-left corner of q_t and q_b are in C, their vertical distance is at most $2\sqrt{2}$. Consequently, there can be at most one other square in Q' that lies in between q_t and q_b, which shows the claim (S).

Now let l_1 be the left vertical tangent of C and let l_2 be its right vertical tangent; see Figure 6.5. We define $Q_l \subseteq Q$ to be the set of squares whose bottom-left corner has distance of at most 1 to l_1. Hence, there is a vertical line that stabs all squares in Q_l. By

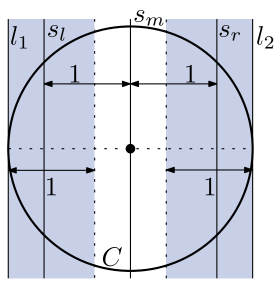


Figure 6.5: Illustration for the proof of Lemma 6.1.

(S) it follows that $|Q_l| \leq 3$. We can analogously define the set $Q_r \subseteq Q$ whose bottom-left corner has distance of at most one to the vertical line l_2. By the same argument it follows that $|Q_r| \leq 3$. Further, the bottom-left corners of the squares $Q_m = Q \setminus \{Q_l, Q_r\}$ are contained in a vertical strip of width $2\sqrt{2} - 2 < 1$. Hence, there is a vertical line that stabs all squares of Q_m and $|Q_m| \leq 3$ follows. We conclude that the set Q contains at most nine squares; in fact, $|Q| \leq 8$ as we show next.

For the sake of contradiction we assume that $|Q| = 9$, i.e., $|Q_l| = |Q_m| = |Q_r| = 3$. We denote the topmost square in Q_l by t_l and the bottommost square by b_l, and define t_r and b_r for Q_r analogously. Further, let s_m be the vertical line through the center of C, let s_l be the vertical line that lies one unit to the left of s_m and let s_r be the vertical line that lies one unit to the right of s_m. Note that the length of the segment of s_l and s_r that is contained in C has length 2. Since the bottom-left corners of t_l and b_l have vertical distance strictly greater than 2, both squares lie to the right of s_l. Hence, t_l and b_l intersect s_m. Analogously, the bottom-left corners of t_r and b_r lie to the left of s_r, and, hence intersect s_r. The line s_m is intersected by two squares of Q_l. By (S) there can be at most one additional square of Q_m that intersects s_m. Thus, there are two squares in Q_m whose anchors lie to the right of s_m. But then they both intersect s_r which itself is already intersected by at least the squares t_r and b_r. This is a contradiction to (S), and concludes the proof. □

Figure 6.6 shows that the bound is tight. Based on Lemma 6.1 we now show that for any label with anchor p there is no point of time $t \in [0, 1]$ for which there can be more than eight active labels whose anchors are within distance $\sqrt{2}$ of p. We call a set $X \subseteq \Psi$ *conflict-free* if it contains no pair of presence intervals that are in conflict. Further, we say that X is in conflict with $I \in \Psi$ if every element of X is in conflict with I, and we say that X contains $t \in [0, 1]$ if every element of X contains t.

Lemma 6.2. *For every $t \in [0, 1]$ and every $I \in \Psi$ any maximum cardinality conflict-free set $X_I(t) \subseteq \Psi$ that is in conflict with I and contains t satisfies $|X_I(t)| \leq 8$.*

Proof. Assume that there is a time t and an interval I such that there is a set $X_I(t)$ that contains more than eight intervals. Let ℓ be the label that corresponds to I. For an interval $I' \in X_I(t)$ to be in conflict with I the anchors of the two corresponding labels

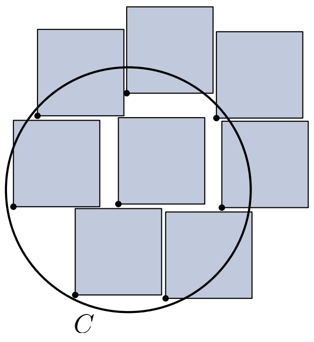


Figure 6.6: Example configuration of eight axis-aligned, non-intersecting, unit-squares with their bottom-left corner inside a circle C with radius $\sqrt{2}$.

have a distance of at most $\sqrt{2}$. Hence, there are $|X_I(t)|$ labels corresponding to the intervals in $X_I(t)$ with anchors of distance at most $\sqrt{2}$ to the anchor of ℓ. By Lemma 6.1 we know that two of these labels overlap. This implies that there is a conflict between the corresponding intervals contained in $X_I(t)$, which is a contradiction. □

With this lemma we can finally obtain the approximation guarantees for GreedyMaxTotal for all activity models.

Theorem 6.2. *Assuming that all labels are unit squares and $w([a,b]) = b - a$, GreedyMaxTotal is a 1/24-, 1/16-, 1/8-approximation for AM1–AM3, respectively, and needs $O(n \log n)$ time for AM1 and $O(n^2)$ time for AM2 and AM3.*

Proof. To show the approximation ratios, we consider an arbitrary step of GreedyMaxTotal in which the presence interval $I = [a,b]_\ell$ is selected from Ψ. Let C_ℓ^I be the set of presence intervals in Ψ that are in conflict with I.

Consider the model AM1. Since I is the longest interval in Ψ when it is chosen, the intervals in C_ℓ^I are completely contained in $J = [a - w(I), b + w(I)]$. As C_ℓ^I contains all presence intervals that are in conflict with I, it is sufficient to consider J to bound the effect of selecting I. Obviously, the interval J is three times as long as I. By Lemma 6.2 we know that for any $X_I(t)$ it holds that $|X_I(t)| \leq 8$ for all $t \in J$. Hence, in an optimal solution there can be at most eight active labels at each point $t \in J$ that are discarded when $[a,b]_\ell$ is selected. Thus, the cost of selecting $[a,b]_\ell$ is at most $3 \cdot 8 \cdot w(I)$.

For AM2 we apply the same arguments, but restrict the interval J to $J = [a, b + w(I)]$, which is only twice as long as I. To see that consider for an interval $[c,d]_{\ell'} \in C_\ell^I$ the prefix $[c,a]$ if it exists. If $[c,a]$ does not exist (because $a < c$), removing $[c,d]_{\ell'}$ from Ψ changes Ψ only in the range of J. If $[c,a]$ exists, then again Ψ is only changed in the range of I, because by definition $[c,d]_{\ell'}$ is shortened to an interval that at least contains $[c,a]$ and is still contained in Ψ. Thus, the cost of selecting I is at most $2 \cdot 8w(I)$.

Analogously, for AM3 we can argue that it is sufficient to consider the interval $J = [a,b]$. By definition of the update operation of GreedyMaxTotal at least the prefix or suffix subinterval of each $[c, d]_{\ell'} \in C_\ell^I$ remains in Ψ that extends beyond I (if such an interval exists). Thus, selecting I influences only the interval J and its cost is at most $8w(I)$. The approximation bounds of 1/24, 1/16, and 1/8 follow immediately.

We use a heap to achieve the time complexity $O(n \log n)$ of GreedyMaxTotal for AM1 since each interval is inserted and removed exactly once. For AM2 and AM3 we use a linear sweep to identify the longest interval contained in Ψ. In each step we need $O(n)$ time to update all intervals in Ψ, and we need a total of $O(n)$ steps. Thus, GreedyMaxTotal needs $O(n^2)$ time in total for AM2 and AM3. □

6.4 Solving k-RestrictedMaxTotal

In this section we prove that unlike GeneralMaxTotal the problem k-Restricted-MaxTotal can actually be solved in polynomial time for AM1 and AM2. We give a detailed description of our algorithm for AM1, and then show how it can be extended to AM2. Note that solving k-RestrictedMaxTotal is related to finding a maximum cardinality k-colorable subset of n intervals in interval graphs. This can be done in polynomial time in both n and k [CL95]. However, we have to consider additional constraints due to conflicts between labels, which makes our problem more difficult. First, we discuss how to solve the case for $k = 1$, then give an algorithm that solves k-RestrictedMaxTotal for $k = 2$, and extend this result recursively to any constant $k > 2$. For AM3 we prove that k-RestrictedMaxTotal is NP-hard for $k \geq 2$. Since the running times of the presented algorithms for AM1 and AM2 are, even for small k, prohibitively expensive in practice, we finally propose an approximation algorithm for k-RestrictedMaxTotal in all three activity models.

6.4.1 An Algorithm for 2-RestrictedMaxTotal in AM1

We start with some definitions before giving the actual algorithm. We assume that the intervals of $\Psi = \{I_1, \ldots, I_n\}$ are sorted in non-decreasing order by their left endpoints; ties are broken arbitrarily. First note that for the case that at most one label can be active at any given point in time ($k = 1$), conflicts between labels do not matter. Thus, it is sufficient to find an independent subset of Ψ of maximum weight. This is equivalent to finding a maximum weight independent set on interval graphs, which can be done in $O(n)$ time using dynamic programming given n sorted intervals [HTC92]. We denote this algorithm by $\mathcal{A}_1$. Let $L_1[I_j]$ be the set of intervals that lie completely to the left of the left endpoint of I_j. Algorithm $\mathcal{A}_1$ basically computes a table $\mathcal{T}_1$ indexed by the intervals in Ψ, where an entry $\mathcal{T}_1[I_j]$ stores the value of a maximum weight independent set Q of $L_1[I_j]$ and a pointer to the rightmost interval in Q.

We call a pair of presence intervals (I_i, I_j), $i < j$, a *separating pair* if I_i and I_j overlap and are not in conflict with each other. Further, a separating pair $\vec{v} = (I_p, I_q)$ is *smaller* than another separating pair $\vec{w} = (I_i, I_j)$ if and only if $p < i$ or $p = i$ and $q < j$.

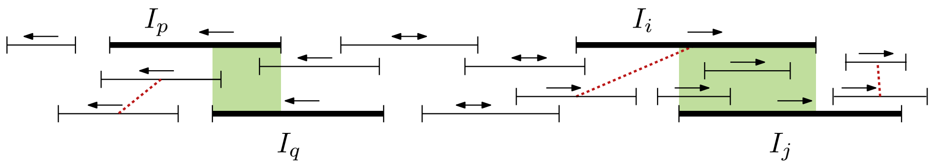


Figure 6.7: Illustration of presence intervals. Intervals that are in conflict are connected by a dotted line. Both (I_i, I_j) and (I_p, I_q) are separating pairs. The intervals of $L_2[i, j]$ ($R_2[p, q]$) are marked by a left (right) arrow.

This induces a total order and we denote the ordered set of all separating pairs by $S_2=\{\vec{v}_1, \ldots, \vec{v}_z\}$. The weight of a separating pair $\vec{v}$ is defined as $w(\vec{v}) = \sum_{I \in \vec{v}} w(I)$.

We observe that a separating pair $\vec{v} = (I_i, I_j)$ contained in a solution of 2-RestrictedMaxTotal splits the set of presence intervals into two independent subsets. Specifically, a left (right) subset $L_2[\vec{v}]$ ($R_2[\vec{v}]$) that contains only intervals which lie completely to the left (right) of the intersection of I_i and I_j and are neither in conflict with I_i nor I_j; see Figure 6.7.

We are now ready to describe our dynamic programming algorithm $\mathcal{A}_2$. For ease of notation we add two dummy separating pairs to S_2. One pair $\vec{v}_0$ with presence intervals strictly to the left of 0 and one pair $\vec{v}_{z+1}$ with presence intervals strictly to the right of 1. Since all original presence intervals are completely contained in $[0, 1]$ every optimal solution contains both dummy separating pairs. Our algorithm computes a one-dimensional table $\mathcal{T}_2$, where for each separating pair $\vec{v}$ there is an entry $\mathcal{T}_2[\vec{v}]$ that stores the value of the optimal solution for $L_2[\vec{v}]$. We compute $\mathcal{T}_2$ from left to right starting with the dummy separating pair $\vec{v}_0$ and initialize $\mathcal{T}_2[\vec{v}_0] = 0$. Then, we recursively define $\mathcal{T}_2[\vec{v}_j]$ for every $\vec{v}_j \in S_2$ as

$$\mathcal{T}_2[\vec{v}_j] = \max_{i<j}\{\mathcal{T}_2[\vec{v}_i] + w(\vec{v}_i) + \mathcal{A}_1(\vec{v}_i, \vec{v}_j) \mid \vec{v}_i \in S_2,\ \vec{v}_i \subseteq L_2[\vec{v}_j],\ \vec{v}_j \subseteq R_2[\vec{v}_i]\}$$

Additionally, we store a backtracking pointer to the predecessor pair that yields the maximum value. In other words, for computing $\mathcal{T}_2[\vec{v}_j]$ we consider all possible direct predecessors $\vec{v}_i \in S_2$ with $i < j$, $\vec{v}_i \cap \vec{v}_j = \emptyset$, and no conflict with $\vec{v}_j$. Each such $\vec{v}_i$ induces a candidate solution whose value is composed of $\mathcal{T}_2[\vec{v}_i]$, $w(\vec{v}_i)$, and the value of an optimal solution of algorithm $\mathcal{A}_1$ for the intervals between $\vec{v}_i$ and $\vec{v}_j$ with $\vec{v}_i$ and $\vec{v}_j$ active.

Since by construction $L_2[\vec{v}_{z+1}] = \Psi \cup \vec{v}_0$, the optimal solution to 2-RestrictedMaxTotal is stored in $\mathcal{T}_2[\vec{v}_{z+1}]$ once $\vec{v}_0$ is removed. To compute a single entry $\mathcal{T}_2[\vec{v}_j]$ our algorithm needs to consider all possible separating pairs preceding $\vec{v}_j$, and for each of them obtain the optimal solution from algorithm $\mathcal{A}_1$ under some additional constraints. For the call $\mathcal{A}_1(\vec{v}_i, \vec{v}_j)$ in the recursive equation above, we distinguish two cases. If the rightmost endpoint of $\vec{v}_i$ is to the left of the leftmost endpoint of $\vec{v}_j$ then we run algorithm $\mathcal{A}_1$ on the set of intervals $L_2[\vec{v}_j] \cap R_2[\vec{v}_i]$ and obtain the value

$\mathcal{A}_1(\vec{v}_i, \vec{v}_j)$. Otherwise, there is an overlap between an interval I_a of $\vec{v}_i$ and an interval I_b of $\vec{v}_j$. Since for $k = 2$ no other interval can cross this overlap, we actually make two calls to $\mathcal{A}_1$, once on the set $R_2[\vec{v}_i] \cap L_2[(I_a, I_b)]$ and once on the set $R_2[(I_a, I_b)] \cap L_2[\vec{v}_j]$. We add both values to obtain $\mathcal{A}_1(\vec{v}_i, \vec{v}_j)$. Since we run algorithm $\mathcal{A}_1$ for each of $O(z)$ separating pairs, the time complexity to compute a single entry of $\mathcal{T}_2$ is $O(nz)$. To compute the whole table the algorithm repeats this step $O(z)$ times, which yields a total time complexity of $O(nz^2)$. Note that the number of separating pairs z is in $O(n^2)$.

We prove the correctness of the algorithm by contradiction. Assume that there exists an instance for which our algorithm does not compute an optimal solution and let OPT be an optimal solution. This means, that there is a smallest separating pair $\vec{v}_j$ for which the entry in $\mathcal{T}_2[\vec{v}_j]$ is less than the value of OPT for $L_2[\vec{v}_j]$. Note that $\vec{v}_j$ cannot be the dummy separating pair $\vec{v}_0$ since $\mathcal{T}_2[\vec{v}_0]$ is trivially correct. Let $\vec{v}_i$ be the rightmost separating pair in OPT that precedes $\vec{v}_j$ and is disjoint from it (possibly $\vec{v}_i = \vec{v}_0$). Since there is no other disjoint separating pair between $\vec{v}_i$ and $\vec{v}_j$ in OPT, all intervals in OPT between $\vec{v}_i$ and $\vec{v}_j$ form a subset of $R_2[\vec{v}_i] \cap L_2[\vec{v}_j]$ that is a valid configuration for $k = 1$. We can obtain an optimal solution for $k = 1$ of the intervals in $R_2[\vec{v}_i] \cap L_2[\vec{v}_j]$ by computing $\mathcal{A}_1(\vec{v}_i, \vec{v}_j)$ as described above. Since, by assumption, $\mathcal{T}_2[\vec{v}_i]$ is optimal, $\mathcal{A}_1$ is correct [HTC92], and our algorithm explicitly considers all possible preceding separating pairs including $\vec{v}_i$, the entry $\mathcal{T}_2[\vec{v}_j]$ are at least as good as OPT for $L_2[\vec{v}_j]$. This is a contradiction and the correctness of $\mathcal{A}_2$ follows.

Theorem 6.3. *Algorithm $\mathcal{A}_2$ solves 2-RestrictedMaxTotal in AM1 in $O(nz^2)$ time and $O(z)$ space, where z is the number of separating pairs in the input instance.*

6.4.2 An Algorithm for k-RestrictedMaxTotal in AM1

In the following, we extend the dynamic programming algorithm $\mathcal{A}_2$ to a general algorithm $\mathcal{A}_k$ for the case $k > 2$. To this end, we extend the definition of separating pairs to separating k-tuples. A *separating k-tuple* $\vec{v}$ is a set of k presence intervals that are not in conflict with each other and that have a non-empty intersection $Y_{\vec{v}} = \bigcap_{I \in \vec{v}} I$. We say a separating k-tuple $\vec{v}$ is *smaller* than a separating k-tuple $\vec{w}$ if $Y_{\vec{v}}$ begins to the left of $Y_{\vec{w}}$. Ties are broken arbitrarily. This lets us define the ordered set $S_k = \{\vec{v}_1, \dots, \vec{v}_z\}$ of all separating k-tuples of a given set of presence intervals. We say a set C of presence intervals is *k-compatible* if no more than k intervals in C intersect at any point and there are no conflicts in C. Two separating k-tuples $\vec{v}$ and $\vec{w}$ are *k-compatible* if they are disjoint and $\vec{v} \cup \vec{w}$ is k-compatible. The definitions of the sets $R_2[\vec{v}]$ and $L_2[\vec{v}]$ extend naturally to the sets $R_k[\vec{v}]$ and $L_k[\vec{v}]$ of all intervals completely to the right (left) of $Y_{\vec{v}}$ and not in conflict with any interval in $\vec{v}$. Now, we recursively define the algorithm $\mathcal{A}_k$ that solves k-RestrictedMaxTotal given a pair of active k-compatible boundary k-tuples. Note that in the recursive definition these boundary tuples may remain k-dimensional even in $\mathcal{A}_{k'}$ for $k' < k$. For $\mathcal{A}_k$ we define as boundary tuples two k-compatible dummy separating k-tuples $\vec{v}_0$ and $\vec{v}_{z+1}$ with all presence intervals strictly to the left of 0 and to the right of 1, respectively. The algorithm fills a one-dimensional table $\mathcal{T}_k$. Similarly to the case $k = 2$, each entry $\mathcal{T}_k[\vec{v}]$ stores the value of the optimal solution for $L_k[\vec{v}]$, i.e., the overall solution can again be obtained from $\mathcal{T}_k[\vec{v}_{z+1}]$. We initialize $\mathcal{T}_k[\vec{v}_0] = 0$. Then, the remaining entries of $\mathcal{T}_k$ can be obtained by computing

$$\begin{aligned}\mathcal{T}_k[\vec{v}_j] = \max_{i<j}\{\mathcal{T}_k[\vec{v}_i] + w(\vec{v}_i) + \mathcal{A}_{k-1}(\tilde{\vec{v}}_i, \tilde{\vec{v}}_j) \mid{} & \vec{v}_i \in S_k,\ \vec{v}_i \subseteq L_k[\vec{v}_j] \cup \vec{v}_0,\\ & \vec{v}_j \subseteq R_k[\vec{v}_i] \cup \vec{v}_{z+1},\\ & \vec{v}_0 \cup \vec{v}_{z+1} \cup \vec{v}_i \cup \vec{v}_j \text{ is } k\text{-compatible}\},\end{aligned}$$

which uses the algorithm $\mathcal{A}_{k-1}$ recursively on a suitable subset of presence intervals between the boundary tuples $\tilde{\vec{v}}_i$ and $\tilde{\vec{v}}_j$. Here $\tilde{\vec{v}}_i$ is defined as the union of the tuple $\vec{v}_i$ and all intervals in $\vec{v}_0 \cup \vec{v}_{z+1}$ that intersect the right endpoint of $Y_{\vec{v}_i}$; analogously $\tilde{\vec{v}}_j$ is defined as the union of the tuple $\vec{v}_j$ and all intervals in $\vec{v}_0 \cup \vec{v}_{z+1}$ that intersect the left endpoint of $Y_{\vec{v}_i}$. This makes sure that in each subinstance all active intervals that are relevant for that particular subinstance are known. Note that by the k-compatibility condition $\tilde{\vec{v}}_i$ and $\tilde{\vec{v}}_j$ contain at most k elements each. In fact, $\mathcal{A}_{k-1}(\tilde{\vec{v}}_i, \tilde{\vec{v}}_j)$ uses $\tilde{\vec{v}}_i$ and $\tilde{\vec{v}}_j$ as boundary k-tuples (and thus does not create dummy boundary tuples) and the set $R_k[\vec{v}_i] \cap L_k[\vec{v}_j]$ as the set of presence intervals from which separating $(k-1)$-tuples can be formed.

Theorem 6.4. *Algorithm $\mathcal{A}_k$ solves k-RestrictedMaxTotal in AM1 in $O(n^{k^2+k-1})$ time and $O(n^k)$ space.*

Proof. We show the correctness of $\mathcal{A}_k$ by induction on k. Theorem 6.3 shows that the statement is true for $k = 2$. Let $k > 2$. Since $\mathcal{A}_k$ only considers solutions where adjacent separating k-tuples are k-compatible with each other and the boundary k-tuples, we cannot produce an invalid solution, i.e., a solution with conflicts or more than k active intervals at any point. We prove the correctness by contradiction. So assume that there is an instance Ψ for which $\mathcal{A}_k$ does not compute an optimal solution and let OPT be an optimal solution. There must be a smallest separating k-tuple $\vec{v}_j$, $j > 0$, for which $\mathcal{T}_k[\vec{v}_j]$ is less than the value of OPT for $L_k[\vec{v}_j]$. Let $\vec{v}_i$, $i < j$ be the rightmost disjoint separating k-tuple in OPT that precedes $\vec{v}_j$ such that the set $\vec{v}_0 \cup \vec{v}_i \cup \vec{v}_j \cup \vec{v}_{z+1}$ is k-compatible. By our assumption $\mathcal{T}_k[\vec{v}_i]$ has the same value as OPT on $L_k[\vec{v}_i]$. For the set of intervals $L_k[\vec{v}_j] \cap R_k[\vec{v}_i]$ there are at most $k-1$ active intervals at any point (otherwise $\vec{v}_i$ is not rightmost). This means that when we run algorithm $\mathcal{A}_{k-1}$ on that instance with the boundary tuples $\tilde{\vec{v}}_i$ and $\tilde{\vec{v}}_j$, i.e., $\vec{v}_i$ and $\vec{v}_j$ enriched by all relevant intervals in $\vec{v}_0 \cup \vec{v}_{z+1}$, we obtain by induction a solution that is at least as good as the restriction of OPT to that instance. Since $\vec{v}_i$ is a valid predecessor

k-tuple for $\vec{v}_j$ the algorithm $\mathcal{A}_k$ considers it. So $\mathcal{T}_k[\vec{v}_j] \geq \mathcal{T}_k[\vec{v}_i] + w(\vec{v}_i) + \mathcal{A}_{k-1}(\tilde{\vec{v}}_i, \tilde{\vec{v}}_j)$, which is at least as good as OPT restricted to $L_k[\vec{v}_j]$. This is a contradiction and proves the correctness.

For proving the time and space complexity let z_i be the number of separating i-tuples in an instance for $1 < i \leq k$. Each z_i is in $O(n^i)$. We again use induction on k. For $\mathcal{A}_2$ Theorem 6.3 yields $O(n^5)$ time and $O(n^2)$ space, which match the bounds to be shown. So let $k > 2$. The table $\mathcal{T}_k$ has $O(z_k) \subseteq O(n^k)$ entries and each of the recursive computations of $\mathcal{A}_{k-1}$ need $O(n^{k-1})$ space by the induction hypothesis. Thus the overall space is dominated by $\mathcal{T}_k$ and the bound follows. Checking whether a separating k-tuple $\vec{v}_i \in S_k$ is a feasible predecessor for a particular $\vec{v}_j$ can easily be done in $O(k^2)$ time, which is dominated by the time to compute $\mathcal{A}_{k-1}(\tilde{\vec{v}}_i, \tilde{\vec{v}}_j)$. So for the running time we observe that each entry in $\mathcal{T}_k$ makes $O(z_k)$ calls to $\mathcal{A}_{k-1}$ and hence the overall running time is indeed $O(n^{2k} \cdot n^{(k-1)^2+(k-1)-1}) = O(n^{k^2+k-1})$. □

6.4.3 Extending the Algorithm for k-RestrictedMaxTotal to AM2

With some modifications and at the expense of another polynomial factor in the running time we can extend algorithm $\mathcal{A}_k$ of the previous section to the activity model AM2, which shows that k-RestrictedMaxTotal in AM2 can still be solved in polynomial time. In the following, we give a sketch of the modifications. The important difference between AM1 and AM2 is that presence intervals can be truncated at their right side if there is an active conflicting witness label causing the truncation. We need two modifications to model this behavior. First, we create for each *original* presence interval $I_i = [a_i, b_i]$ in Ψ at most n prefix intervals $I_i^j = [a_i, c_{ij}]$, where c_{ij} is the start of the first conflict between I_i and $I_j \in \Psi$. Each interval I_i^j inherits the conflicts of I_i that intersect I_i^j. We obtain a modified set of presence intervals $\Psi' = \Psi \cup \{I_i^j \mid I_i, I_j \in \Psi \text{ and } I_i, I_j \text{ in conflict}\}$ of size $O(n^2)$. We create mutual conflicts among all intervals that are prefixes of the same original interval. This will enforce that at most one of them is active. We still have to take care that a truncated interval I_i^j can only be active if I_j (or a prefix of I_j) is active at c_{ij} as a witness.

In order to achieve this we instantiate the algorithm $\mathcal{A}_{k'}$ for every $k' \leq k$ not only with its two boundary k-tuples $\tilde{\vec{v}}_0$ and $\tilde{\vec{v}}_{z+1}$ but also with a set W of at most k witness intervals that are k-compatible and must be made active at some stage of the algorithm. In a valid solution we have $W \subseteq L_{k'}[\vec{v}] \cup \vec{v}$ for the leftmost separating k'-tuple $\vec{v}$, since otherwise more than k' intervals are active in $Y_{\vec{v}}$. However, the truncated intervals in $\vec{v}$ themselves define a family of $O(n^{k'})$ possible witness sets $W(\vec{v})$ to be respected to the right of $\vec{v}$. So when we compute the table entry for a separating k'-tuple $\vec{v}_j$ and consider a particular predecessor k'-tuple $\vec{v}_i$ we must in fact iterate over all possible witness sets $W(\vec{v}_i)$ as well. We need to make sure that $\vec{v}_j$ is *$W(\vec{v}_i)$-compatible*, i.e., $\vec{v}_j \cup W(\vec{v}_i)$ is k-compatible and $W(\vec{v}_i) \subseteq L_{k'}[\vec{v}_j] \cup \vec{v}_j$. For the recursive call to $\mathcal{A}_{k'-1}(\tilde{\vec{v}}_i, \tilde{\vec{v}}_j)$ the initial witness set W' consists of $W(\vec{v}_i) \setminus \vec{v}_j$, i.e., those witness intervals of $W(\vec{v}_i)$ that are not part of $\vec{v}_j$.

The increase in running time is caused by dealing with $O(n^2)$ intervals in Ψ' and by the fact that instead of one call to $\mathcal{A}_{k-1}(\tilde{\vec{v}}_i, \tilde{\vec{v}}_j)$ in the computation of table $\mathcal{T}_k$ we make $O(n^k)$ calls, one for each possible witness set of $\vec{v}_i$. By an inductive argument one can show that the running time is in $O(n^{3k^2+2k})$.

Theorem 6.5. *k-RestrictedMaxTotal in AM2 can be solved in polynomial time.*

6.4.4 Complexity of k-RestrictedMaxTotal in AM3

In this section we discuss the complexity of k-RestrictedMaxTotal with respect to AM3. For $k = 1$ the problem is equivalent with finding a maximum weighted independent set on interval graphs, which can be done in linear time [HTC92].

We show that the according decision problem, which is defined as follows, is NP-complete for $k \geq 2$ and the combinatorial variant of the problem. The complexity of the geometric variant is an open problem.

Problem 6.3 (**k**-RestrictedMaxTotalDecision).
Given: *Instance $\mathcal{I} = (L, \Psi, C)$, $k \in \mathbb{N}$ and $W \in \mathbb{R}^+$.*
Question: *Is there an activity Φ for $\mathcal{I}$ such that $\sum_{[a,b]_\ell \in \Phi} w([a,b]_\ell) \geq W$ and at most k labels are active at the same time?*

In the following, we require that the activity Φ also satisfies AM3. It is easy to see that the decision problem is NP-complete. For each label $\ell \in L$ we guess intervals that are contained in the presence intervals of ℓ and whose beginnings and ends coincide with the beginnings and ends of presence and conflict intervals in $\Psi \cup C$. We further check, whether the intervals are valid activity intervals with respect to AM3, their weights sum up at least to W and whether at most k labels are active at the same time. Obviously, we can check this in polynomial time in the size of the input.

In the remainder of this section we show that the decision problem is also NP-hard for $k \geq 2$, which implies that the according optimization problem is NP-hard as well. We do a reduction from the NP-complete problem 3-Partition. For the convenience of the reader, we repeat the definition given by Garey and Johnson[GJ79].

Problem 6.4 (3-Partition).
Given: *Set X of $3m$ elements, a* bound *$B \in \mathbb{Z}^+$, and a* size *$s(x) \in \mathbb{Z}^+$ for each element $x \in X$ such that $\frac{B}{4} \leq s(x) \leq \frac{B}{2}$ and $\sum_{x \in X} s(x) = mB$.*
Question: *Can X be partitioned into m disjoint sets $X_1, X_2, \ldots, X_m$ such that $\sum_{x \in X_i} s(x) = B$ for all i with $1 \leq i \leq m$?*

Note that each X_i must contain exactly three elements. We first present a construction that transforms a given instance $\mathcal{I}$ of 3-Partition into an instance $\mathcal{I}'$

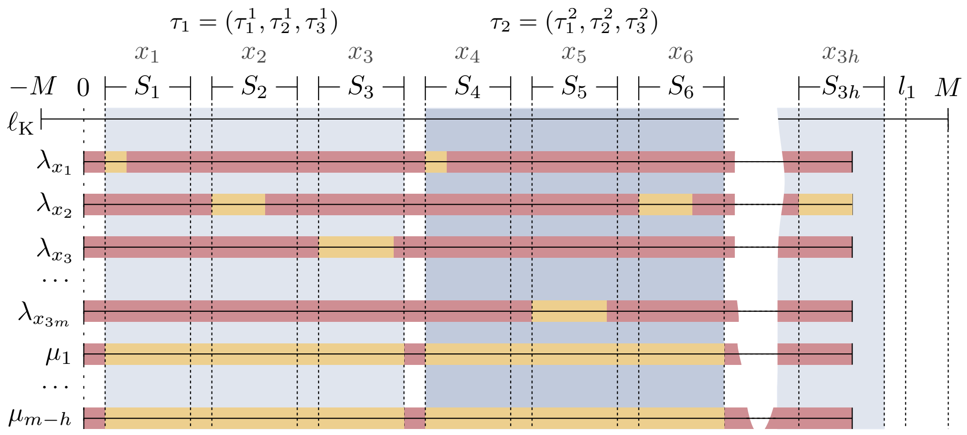


Figure 6.8: Illustration of the construction used for the reduction from 3-Partition to k-RestrictedMaxTotal. Times for which a label $\ell \in L_\lambda \cup L_\mu$ is in conflict with ℓ_K are marked red. For each label $\lambda_x \in L_\lambda$ the times when it can possibly be active are marked orange.

of k-RestrictedMaxTotal. Afterwards we show that $\mathcal{I}$ is an *yes*-instance of 3-Partition if and only if $\mathcal{I}'$ is an *yes*-instance of k-RestrictedMaxTotal with respect to AM3.

The construction is illustrated in Figure 6.8. Let $s_{max} = \max_{x \in X} s(x)$. We first construct every triplet $\tau \subseteq X$ such that the elements in τ are pairwise different and $\sum_{x \in \tau} s(x) = B$. Let $T = \{\tau_1, \dots, \tau_h\}$ denote the set of those triplets. For each triplet $\tau_i \in T$ we fix an arbitrary order of its elements, and denote its j-th element by τ_i^j (with $1 \leq j \leq 3$). We define

$$x_1 = \tau_1^1,\ x_2 = \tau_1^2,\ x_3 = \tau_1^3,\ x_4 = \tau_2^1,\ x_5 = \tau_2^2, \dots, x_{3h} = \tau_h^3.$$

For each element $x \in X$ we create one label λ_x with presence interval

$$I = [0, 3h(s_{max} + 1) + 1].$$

Let L_λ denote the set of those labels and let l_1 denote the length of those intervals. The interval I can be partitioned into $3h$ intervals $\mathcal{S}$ of length s_{max} and $3h + 1$ intervals $\mathcal{U}$ of unit length such that the intervals in $\mathcal{S}$ and $\mathcal{U}$ alternate, starting with an interval of $\mathcal{U}$; see Figure 6.8. We call an interval in $\mathcal{S}$ a *slot*. We denote the slots by $S_1 = [a_1, b_1], \dots, S_{3h} = [a_{3h}, b_{3h}]$ in increasing order, i.e., $a_i < a_j$ for $i < j$. We say that the slots S_{3i-2}, S_{3i-1} and S_{3i} *belong to* the triplet τ_i ($1 \leq i \leq h$).

Thus, for each x_i we have a slot S_i with $1 \leq i \leq 3h$. In our construction we will enforce that a label $\lambda_x \in L_\lambda$ can only be active during a slot S_i ($1 \leq i \leq 3h$) if $x_i = x$. More precisely, we define

$$S_{x,i} = \begin{cases} [a_i, a_i + s(x)], & x_i = x \\ \emptyset, & \text{otherwise,} \end{cases}$$

Thus, $S_{x,i} \subseteq S_i$. We will enforce that λ_x can only be active during $S_{x,i}$. To that end we introduce a single label ℓ_K with presence interval

$$K = [-M, l_1 + M],$$

where we choose $M > 3l_1$. We denote the length of K by l_2 and note that $l_2 \geq 2M$.

For each $x \in X$ we introduce conflicts between λ_x and ℓ_K such that λ_x and ℓ_K are *not* in conflict at time t if and only if there is an i with $1 \leq i \leq 3h$ and $t \in S_{x,i}$. Put differently, λ_x and ℓ_K are in conflict during the time expressed by

$$I \setminus \bigcup_{i=1}^{3h} S_{x,i}$$

Assume that ℓ_K is active during the complete interval $[0, l_1]$, then any label λ_x with $x \in X$ can only be active during $\bigcup_{i=1}^{3h} S_{x,i}$. Since the slots are disjoint, since $S_{x_i} \subseteq S_i$, and since each label can only be active once per presence interval, each label λ_x can only be active during at most one slot S_i, but not outside of a slot. Further, the element x must be contained in the according triplet τ_j with $j = \lceil \frac{i}{3} \rceil$.

Moreover, we introduce $h - m$ labels with presence interval I. We denote the set of these labels by L_μ. We define that any label $\mu \in L_\mu$ and ℓ_K are in conflict during the time expressed by

$$I \setminus \bigcup_{j=1}^{h} [a_{3j-2}, b_{3j}]$$

Thus, μ is not in conflict with λ_k during any interval $T_i = [a_{3i-2}, b_{3i}]$ (with $1 \leq i \leq h$), which spans the slots S_{3i-2}, S_{3i-1}, S_{3i} of the triplet τ_i. We note that the length of T_i is $l_3 = 3s_{max} + 2$. We further define that the labels in $L_\lambda \cup L_\mu$ are pairwise in conflict during the complete interval I, which implies that at most two labels can be active at any time, namely ℓ_K and at most one label of $L_\lambda \cup L_\mu$. Finally, for any constructed label ℓ we set $w_\ell = 1$ and define

$$W = l_2 + \sum_{x \in X} s(x) + (h - m) \cdot l_3$$

Hence, we have $w([a, b]_\ell) = (b - a)$ for any presence/activity interval of ℓ.

Altogether, we obtain the instance $\mathcal{I}' = (L = L_\lambda \cup \{\ell_K\} \cup L_\mu, \Psi, C, W)$, which can obviously be constructed in polynomial time with respect to the given 3-Partition instance $\mathcal{I}$.

Lemma 6.3. *Let $\mathcal{I}$ be an instance of 3-Partition and let $\mathcal{I}'$ be an instance of k-RestrictedMaxTotalDecision ($k \geq 2$) with respect to AM3 constructed from $\mathcal{I}$ as described. The instance $\mathcal{I}$ is a* yes*-instance if and only if $\mathcal{I}'$ is a* yes*-instance.*

Proof. First assume that $\mathcal{I}$ is a *yes*-instance of 3-Partition. Let $T' \subseteq T$ be the set of the m triplets such that each element $x \in X$ is contained in exactly one triplet of T' and each triplet of T' sums up to B. We now construct an activity Φ for $\mathcal{I}'$ with weight $W_\Phi \geq W$. To that end we first set ℓ_K active for the interval K. Thus, $W_\Phi \geq l_2$.

Further, for each triplet $\tau \in T'$ and each element $x \in \tau$ we add the slot $S_{x,i}$ to Φ, where $1 \leq i \leq 3h$, $x = x_i$ and $S_{x,i}$ belongs to τ. By construction λ_x is not in a conflict with ℓ_K during $S_{x,i}$, but directly before and after that interval. Since K is completely contained in Φ, the label ℓ_K is also active directly before and after that interval, which altogether satisfies AM3. Further, only two labels are active at the same time. This follows directly from the fact that the slots are disjoint and we set for each slot at most two labels active, namely λ_x and ℓ_K. Since each added interval $S_{x,i}$ has length $s(x)$ and each x belongs to exactly one triplet $\tau \in T'$, the activity Φ now has weight $W_\Phi = l_2 + \sum_{x \in X} s(x)$.

There are $h - m$ triplets $\tau_{i_1}, \dots, \tau_{i_{h-m}}$ in $T \setminus T'$. For each such triplet τ_{i_j} we know by construction that no label $\lambda_x \in L_\lambda$ is active during T_{i_j}. Hence, we set the label μ_j active during the interval T_{i_j}. Since μ_j is in conflict with ℓ_K directly before and after T_{i_j}, the model AM3 is satisfied. Further, since $T_{i_1}, \dots, T_{i_{h-m}}$ are pairwise disjoint, at most two labels are active at the same time. In total we obtain a valid activity Φ with weight

$$W_\Phi = l_2 + \sum_{x \in X} s(x) + (h - m) \cdot l_3 = W.$$

Now assume that $\mathcal{I}'$ is a *yes*-instance of k-RestrictedMaxTotal ($k \geq 2$) with respect to AM3, i.e., we are given an activity Φ for $\mathcal{I}'$ with $W_\Phi \geq W$. We show how to construct a valid solution for $\mathcal{I}$. Recall that since the labels in $L_\lambda \cup L_\mu$ are pairwise in conflict for the complete interval I, at most two labels can be active at the same time, namely ℓ_K and one label of $L_\lambda \cup L_\mu$.

We now argue that ℓ_K is active for the complete interval $[0, l_1]$. If this was not the case, then the prefix $[-M, 0]$ or the suffix $[l_1, l_1 + M]$ of K cannot belong to the activity of ℓ_K. Further, for the interval $[0, l_1]$ we obtain less than $\frac{2}{3}M$ weight in total, because at most two labels can be active at the same time and $l_1 < \frac{M}{3}$. Thus, we obtain $W_\Phi < M + \frac{2}{3}M < 2M$. On the other hand, because of $l_2 \geq 2M$ and $W_\Phi \geq W$, we have $W_\Phi \geq 2M$, which is a contradiction. On that account, the label ℓ_K is active during the complete interval $[0, l_1]$. It is even active for the complete interval K to sustain the requirements of AM3. As reasoned previously, the activity interval of any label $\ell \in L_\lambda$ must be thus contained in one of the slots $S_1, \dots, S_{3h}$; otherwise there would be an unresolved conflict between ℓ and ℓ_K. Similarly, the activity interval of any label $\ell \in L_\mu$ must be contained in one of the intervals $T_1, \dots, T_h$.

By the construction of the conflicts, a label $\lambda_x \in L_\lambda$ can be active for at most $s(x)$ time and a label $\mu_i \in L_\mu$ can be active for at most l_3 time. Due to $W_\Phi \geq W$, each label $\lambda_x \in L_x$ is therefore active for exactly $s(x)$ time and, consequently, there is a slot $S_{x,i}$ with $1 \leq i \leq 3h$ that is the activity interval of λ_x. Similarly each label $\mu \in L_\mu$ is active for exactly l_3 time and, consequently, there is an interval T_i with $1 \leq i \leq h$ that is the activity interval of μ.

More precisely, there are m triplets $\tau_{i_1}, \ldots, \tau_{i_m}$ such that the activity intervals of the labels in L_λ are contained in $\bigcup_{j=1}^{m} T_{i_j}$. To see that, consider any label $\mu \in L_\mu$. By the previous reasoning there is a triplet τ_i with $1 \leq i \leq h$ such that the activity interval T_i of μ contains $S_{3i-2}, S_{3i-1}, S_{3i}$. This implies that no label $\lambda_x \in L_\lambda$ with $x \in \tau_i$ can be active during T_i. Since no two labels $\mu, \mu' \in L_\mu$ can be active during the same interval T_i and $|L_\mu| = h - m$, the m intervals $T_{i_1}, \ldots, T_{i_m}$ remain containing the activity intervals of the labels in L_λ.

Those triplets $\tau_{i_1}, \ldots, \tau_{i_m}$ form the desired solution for $\mathcal{I}$: Since each label $\lambda_x \in L_\lambda$ is active for exactly one interval, each element $x \in X$ is contained in exactly one triplet τ_{i_j} with $1 \leq j \leq m$. Further, we have $\sum_{x \in \tau_{i_j}} s(x) = B$, which concludes the proof. □

The lemma directly implies that the problem k-RestrictedMaxTotal is NP-hard for $k \geq 2$ and AM3. We summarize the results of this section in the following theorem.

Theorem 6.6. *For $k = 1$ the problem k-RestrictedMaxTotal can be solved in linear time in AM3, while it is NP-hard for $k \geq 2$.*

6.4.5 Approximation of k-RestrictedMaxTotal

Since the running times of our algorithms for k-RestrictedMaxTotal are, even for small k, prohibitively expensive in practice, we propose an approximation algorithm for k-RestrictedMaxTotal based on GreedyMaxTotal for all activity models.

Our algorithm GreedyRestrictedMaxTotal is a simple extension of GreedyMaxTotal. Recall that GreedyMaxTotal greedily removes the longest interval I from Ψ and adds it to the set Φ that contains the active intervals of the solution. Then, it updates all intervals contained in Ψ that are in conflict with I. This process is repeated until Ψ is empty. For approximating k-RestrictedMaxTotal we need to ensure that there is no point in time t that is contained in more than k intervals in Φ. We call intervals which we cannot add to Φ without violating this property *invalid*.

Our modification of GreedyMaxTotal is as follows. After adding an interval I to Φ and handling conflicts as before, we remove intervals from Ψ that became invalid. We say that we *ensure that I is valid*. Note that we cannot shorten those intervals because then we could not ensure that adding an interval from Ψ to Φ is valid according to our model.

In order to prove approximation ratios we first introduce the following lemma that describes the structure of a solution of k-RestrictedMaxTotal.

Lemma 6.4. *Let S be a set of intervals such that there is no number that is contained in more than k intervals from S. Then, there is a partition of S into k sets $M_1, \dots, M_k$, such that no two intersecting intervals are in the same set M_i.*

Proof. Let $I_1, \dots, I_m$ be all intervals of S sorted by their left endpoints in non-decreasing order. In the following, we describe how to construct the partition.

We start with empty sets $M_1, \dots, M_k$. First, add I_1 to M_1. Assume that the first $i-1$ intervals have been added to the sets $M_1, \dots, M_k$. We describe how to add I_i. If there is an empty set M_j then, we simply add I_i to M_j. Otherwise, let $I_{i_1}, \dots, I_{i_k}$ be the rightmost intervals in the sets $M_1, \dots, M_k$, respectively. We denote the set containing those intervals by R. Let $\mathcal{I} = \bigcap_{I \in R} I$. If $\mathcal{I}$ is not empty then, due to the order of the intervals, the interval I_i cannot begin to the left of $\mathcal{I}$. It also cannot begin in $\mathcal{I}$ because otherwise there would be a number that is contained in $k+1$ intervals in S. Let $M_x, 1 \leq x \leq k$ be the set that contains the interval $I \in R$ with leftmost right endpoint among the intervals in R. Since I_i lies completely to the right of $\mathcal{I}$ it must also lie completely to the right of I. Thus we can assign I_i to the set M_x without introducing intersections. If $\mathcal{I}$ is empty, then there must be an interval $I \in R$ with right endpoint to the left of another $I' \in R$. Let $M_x, 1 \leq x \leq k$ be the set that contains the interval I. Due to the order of the intervals the interval I_i lies completely to the right of I and hence we assign I_i to M_x without introducing intersections. This concludes the proof. □

With this lemma we now can prove the following theorem that makes a statement about the approximation ratio of GreedyRestrictedMaxTotal

Theorem 6.7. *Assuming that all labels are unit squares and $w([a,b]) = b - a$, Greedy-RestrictedMaxTotal is a $1/\min\{3+3k, 27\}$, $1/\min\{3+2k, 19\}$, $1/\min\{3+k, 11\}$-approximation for AM1–AM3, respectively, and needs $O(n^2)$ time.*

Proof. We begin by proving its correctness and then we show its time complexity.

Consider the step in which we add an interval $I = [a,b]$ to Φ, and let $J = [a - w(I), b + w(I)]$. Let $\mathcal{L}$ be a fixed, but arbitrary optimal solution. If $I \in \mathcal{L}$, there is no lost weight compared to the optimal solution when choosing I.

Thus, assume that $I \notin \mathcal{L}$. Let $C(I) \subseteq \mathcal{L}$ be the set of intervals that are in conflict with I. Identically to the proof of Theorem 6.2 we can argue that $w(C(I)) = \sum_{I \in C} w(I) \leq (4 - X) \cdot 8 \cdot w(I)$ considering activity model AMX with $X \in \{1,2,3\}$.

We now show that at most $3w(I)$ weight of the optimal solution is lost when ensuring that I is valid.

By Lemma 6.4 we can partition $\mathcal{L}$ into k sets $M_1, \dots, M_k$ such that no two intersecting intervals are in the same set M_i. If I is in $\mathcal{L}$, then we do not lose any weight compared to the optimal solution. Hence, assume that I is not in $\mathcal{L}$. Take any M_i, $1 \leq i \leq k$, remove all intervals of $\mathcal{L} \setminus \Phi$ that intersect I. We denote the set of removed intervals by R. In the following, we bound the cost of removing the intervals in R. If there are intervals in R that are longer than I, then we have already accounted for them in previous steps. This relies on the fact that we consider the intervals sorted by their length in non-ascending order, and, hence if those longer intervals are not in Φ, we must have removed them in an earlier step. Thus, we only need to bound the length of intervals in R which have length at most $w(I)$. Those intervals must lie in J, and due to the definition of M_i they must be disjoint. Hence, the cost is bounded by $3w(I)$.

Altogether choosing I causes that at most $3w(I) + (4 - X) \cdot 8 \cdot w(I)$ weight is lost compared to the optimal solution considering activity model AMX with $X \in \{1, 2, 3\}$. Finally, this yields an approximation factor of 1/27, 1/19, 1/11 for AM1-3, respectively.

For $k < 8$ we can improve $w(C(I)) \leq (4 - X) \cdot 8 \cdot w(I)$ to $w(C(I)) \leq (4 - X) \cdot k \cdot w(I)$ considering activity model AMX with $X \in \{1, 2, 3\}$ because we know that I cannot be in conflict with more than k intervals of the optimal solution. Thus, we can bound the loss of choosing I by $3w(I) + (4 - X) \cdot k \cdot w(I)$. In total this yields the claimed approximation ratios for the three activity models.

Finally, we argue the correctness of the claimed running time of $O(n^2)$. Since the worst-case running time of GreedyMaxTotal is $O(n^2)$ we only need to argue that we can delete those intervals from Φ, which are not valid anymore, in $O(n)$ time per step. To do this we simply sort the intervals in Ψ in non-decreasing order by their left-endpoint. We also maintain Φ in the same way. Then, we can check for non-valid intervals with a simple linear sweep over Ψ and Φ. Hence, each iteration of the algorithm requires $O(n)$ time, which yields a total running time of $O(n^2)$. □

6.5 Conclusions

In this chapter, we introduced a temporal model for dynamic map labeling that satisfies the consistency criteria demanded by Been et al. [BDY06], even in a stronger sense, where each activity change of a label must be explainable to the user by some witness label. Our model transforms the geometric information specified by the motion of the camera as well as the labels into the two combinatorial problems GeneralMaxTotal and k-RestrictedMaxTotal that are expressed in terms of presence and conflict intervals. Thus, our algorithms apply to any dynamic labeling problem that can be transformed into such an interval-based problem.

We showed that GeneralMaxTotal is NP-complete and W[1]-hard and presented constant-factor approximation algorithms for our three different activity models. The problem k-RestrictedMaxTotal, where at most k labels can be visible at any time, can be solved in polynomial time $O(n^{f(k)})$ in activity models AM1 and AM2 for any fixed k, where f is a polynomial function. Due to the W[1]-hardness of GeneralMaxTotal we cannot expect to find better results for the running times, apart from improving upon the function f. Further, we proved NP-hardness for k-RestrictedMaxTotal in AM3 for $k \geq 2$. We also presented an $O(n^2)$-time approximation algorithm for k-RestrictedMaxTotal in all three activity models.

The analysis of the approximation algorithms for both k-RestrictedMaxTotal and GeneralMaxTotal significantly relies on the assumption that labels are unit squares. Thus, the question arises whether constant-factor approximations exist when this assumption is dropped or softened, e.g., to labels of unit-height. To answer this question we think that deeper insights into the structure of conflicts are necessary, e.g., does the geometric information based on viewport, its motion and labels imply a useful structure on the induced label conflict graph?

7 Temporal Map Labeling: An Algorithmic Framework

Abstract. Through extensive experiments on OpenStreetMap data, we evaluate the model for temporal map labeling presented in the previous chapter. We use algorithms of varying complexity in a case study for navigation systems. Our experiments show that even simple (and thus, fast) algorithms achieve near-optimal solutions in our model with respect to an intuitive objective function. In order to obtain optimal solutions, we use integer linear programming formulations for GeneralMaxTotal and k-RestrictedMaxTotal.

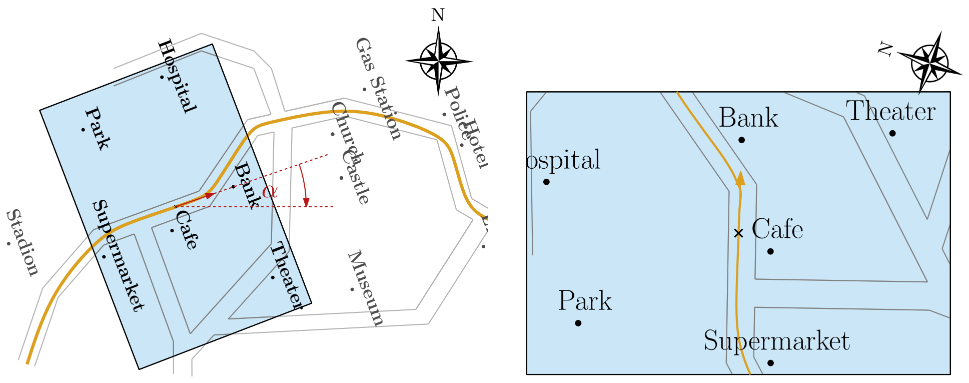
This chapter is based on and partly taken from joint work with Andreas Gemsa, Lukas Barth, Martin Nöllenburg and Darren Strash [GNN13a, GNN13b, Bar+16].

7.1 Introduction

In this chapter, we build upon our previous work introduced in Chapter 6, present more sophisticated heuristic algorithms, and provide an extensive experimental evaluation of our proposed temporal labeling models and algorithms in a case study for navigation systems. Our experiments illustrate the usefulness of our models for this application, and further show the strength of each algorithm under each model. Ultimately, our experiments show that simple but fast algorithms achieve near-optimal solutions for the optimization problems—which is very encouraging, given the hardness results of the previous chapter.

In Section 7.2, we first introduce a simple but powerful workflow consisting of two phases. In the first phase, the input problem is transformed into a temporal labeling instance of our model. Then in the second phase we solve GeneralMaxTotal and k-RestrictedMaxTotal on that instance. We present algorithms solving the problems heuristically as well as exact integer linear programming formulations. In Section 7.3, we experimentally evaluate our approach considering the application of navigation systems. Utilizing the integer linear programming formulations, we obtain optimal solutions for a set of test instances. We use them to show that our heuristics yield near-optimal solutions, while they are fast and simple enough to be deployed in practice.

In the remainder of the chapter, we use the notation and model of Chapter 6. In particular, we assume that we are given a set $O = \{o_1, \ldots, o_n\}$ of objects in a scene over a given time span $\mathcal{T} = [0, 1]$. Further, using the metaphor of a camera, the objects are projected onto an infinite plane P such that we can only see a restricted section V



(a) Overall view. **(b)** Viewport.

Figure 7.1: Trajectory-based labeling. (a) Viewport moves and aligns along a given trajectory (fat orange line). Labels align to the viewport. (b) The user's view on the scene.

of P, where V models the *viewport* of the camera. For a detailed introduction to the model see Chapter 6.

7.2 Workflow

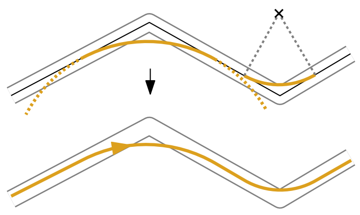
In this section we describe a simple but flexible workflow for temporal labeling problems. This workflow consists of two phases. In the first phase a concrete geometric labeling problem is transformed into an abstract temporal labeling instance $I = (L, \Psi, C)$, where $L = \{\ell_1, \ldots, \ell_n\}$ is the set of labels (ℓ_i is the label of object o_i), Ψ is the set of presence intervals and C is the set of conflict intervals.

This step critically depends on the concrete geometric model of the given temporal labeling problem. Here, we consider the application of a car navigation system; other labeling problems, such as labeling moving entities, can be handled similarly. In the second phase, either GeneralMaxTotal or k-RestrictedMaxTotal is solved for the output instance I from the first phase. We now describe these two phases in greater detail.

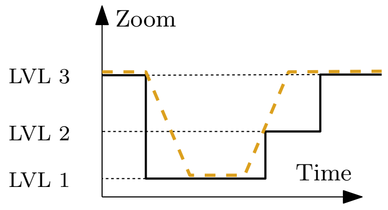
7.2.1 Phase 1 – Transformation into Intervals

This phase depends on the specific labeling problem given. It transforms the input for a particular geometric setting into a temporal labeling instance that can then be handled independently from the geometry.

Example: Navigation Systems. For our experiments we consider the use case of car navigation systems. In this use case the viewport of the map moves along a selected



(a) Smoothing corners.



(b) Interpolating zoom.

Figure 7.2: Trajectory. (a) The corners of the selected route (black polyline) are smoothed by circular arcs obtaining a continuous and differentiable trajectory (orange curve). (b) The black rectilinear curve shows the zoom levels assigned to the underlying roads over time. The orange dashed line illustrates the interpolated actual zooming of the viewport.

route to the journey's destination such that the camera is perpendicular to the map; that is, the user of the navigation system observes the map in aerial perspective such that at any time the viewport has a certain position, rotation, and scale; see Figure 7.1.

We model the viewport as an arbitrarily oriented rectangle R that defines the currently visible part of the map on the screen. The viewport follows a trajectory that we model as a continuous differentiable function $\tau\colon \mathcal{T} \to \mathbb{R}^2$.

In our setting, we obtain τ from a polyline describing the selected route by *smoothing* the polyline's corners by circular arcs; see Figure 7.2(a). Thus, τ is described by a sequence of line segments and circular arcs.

The viewport is described by a function $V\colon \mathcal{T} \to \mathbb{R}^2 \times [0, 2\pi] \times [0, 1]$. The interpretation of $V(t) = (c(t), \alpha(t), z(t))$ is that at time t the center of R is located at $c(t)$, R is rotated clockwise by the angle $\alpha(t)$ relatively to a north base line of the map, and R is scaled by the factor $z(t)$. We call $z(t)$ the *zoom* of V at time t. Since R moves along τ we define $c(t) = \tau(t)$. To avoid distracting changes of the map, we assume that the viewport does not both rotate and zoom at the same time. More precisely, we are given a finite set $\mathcal{Z}$ of zoom levels at which the viewport is allowed to rotate. Hence, when the camera zooms, the trajectory must form a straight line for that particular period of time.

The objects in O describe points of interests and are fixed on the map. We model a label ℓ of an object $o \in O$ as a rectangle on the plane P that is anchored at the projection of o onto P with the midpoint of its bottom side. It does not change its size on the screen over time. To ensure good readability, the labels are always aligned with the viewport axes as the viewport changes its orientation (i.e., they rotate around their anchors by the same angle $\alpha(t)$); see Figure 7.1.

For each label we compute the time events when it enters or leaves the viewport, and when it starts and stops overlapping another label. Since rotation and zooming are temporally separated, those operations can be considered independently. Computing the time events for rotations requires an intricate geometric analysis, which is described

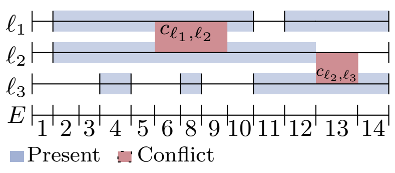


Figure 7.3: The presence and conflict intervals are split into disjoint atomic segments by their events E.

in [Nie12]. For changing from one zoom level to another, we do not allow instantaneous changing of zoom levels, but instead we linearly interpolate the scale of the map between both zoom levels, as in Figure 7.2(b). (In our experiments we further enforce a minimum duration between two changes of zoom levels to avoid oscillation effects; see Section 7.3.) Under these conditions, time events for zooming can be computed by detecting collisions among linearly moving objects.

The computed time events directly translate into presence and conflict intervals of the labels. Hence, we obtain the temporal labeling instance $I = (L, \Psi, C)$.

Other Scenarios. Our model is not restricted to labels of point features, but it also can be applied to labels of other features such as line and area features. For example, one could pre-compute a label placement for roads and combine the road labels with labels for point features by computing all temporal conflict events. Thus, we again obtain a temporal labeling instance $I = (L, \Psi, C)$ describing the setting. By pre-selecting active intervals for certain labels, we can further enforce that they are definitely active at the selected times. In the same manner we can ensure that labels do not overlap certain important map features. Finally, we do not require the labeled objects to be fixed, but they may also move. As long as the start and end times of label presence- and conflict intervals can be determined in advance, they can be represented in our model. Depending on the setting, this may involve non-trivial geometrical computations, but once the transformation is done, the different scenarios are treated equally.

7.2.2 Phase 2 – Resolving Conflicts

In the second phase, we compute the activity intervals for all labels. We present optimal approaches as well as efficient heuristics for solving GeneralMaxTotal and k-RestrictedMaxTotal on $I = (L, \Psi, C)$. Recall from Chapter 6 that we say that an activity set Φ is *valid* if

(R1) for each activity interval $I_\ell \in \Phi$ there is a presence interval $I'_\ell \in \Psi$ with $I_\ell \subseteq I'_\ell$,
(R2) for each presence interval $I_\ell \in \Psi$ there is at most one activity interval $I'_\ell \in \Phi$ with $I'_\ell \subseteq I_\ell$, and
(R3) no two activity intervals of Φ are in conflict.

Integer Linear Programming. In order to provide upper bounds for the evaluation of our labeling algorithms, we have developed integer linear programming (ILP) formulations for the model of Chapter 6 that solves GeneralMaxTotal and k-RestrictedMaxTotal optimally. Finding an optimal solution for an ILP model is NP-hard in general. However, it turns out that in practice we can apply specialized solvers to find optimal solutions for reasonably sized instances in acceptable time; see Section 7.3 for details. Hence, this ILP-based method provides a simple and generic way to produce optimal solutions. We call this approach Ilp.

We present the formulation for the most involved model AM3 in GeneralMaxTotal and then argue how to adapt it to the simpler models AM1 and AM2. Finally, we explain how to adapt the formulation for k-RestrictedMaxTotal. We assume that all presence and conflict intervals are sub-intervals of $[0,1]$.

We define E to be the totally ordered set of the endpoints of all presence and all conflict intervals and include 0 and 1; see Figure 7.3. We call each interval $[c,d]$ between two consecutive elements c and d in E an *atomic segment* and denote the i-th atomic segment of E by $E(i)$. Further, let $X(\ell,i)$ be the set of labels that are in conflict with ℓ during $E(i-1)$, but not during $E(i)$, i.e., the conflicts end with $E(i-1)$. Analogously, let $Y(\ell,i)$ be the set of labels that are in conflict with ℓ during $E(i+1)$, but not during $E(i)$, i.e., the conflicts begin with $E(i+1)$. For each label ℓ we introduce three binary variables $b_i, x_i, e_i \in \{0,1\}$ and the following constraints.

$$b_i^\ell = x_i^\ell = e_i^\ell = 0 \qquad \forall 1 \le i \le |E| \text{ s.t. } \forall [c,d] \in \Psi_\ell : E(i) \cap [c,d] = \emptyset \tag{7.1}$$

$$\sum_{j\in J} b_j^\ell \le 1 \text{ and } \sum_{j\in J} e_j^\ell \le 1 \qquad \forall [c,d] \in \Psi_\ell \text{ where } J = \{j \mid E(j) \subseteq [c,d]\} \tag{7.2}$$

$$x_i^\ell + x_i^{\ell'} \le 1 \qquad \forall 1 \le i \le |E| \; \forall [c,d]_{\ell,\ell'} \in C : \; E(i) \subseteq [c,d] \tag{7.3}$$

$$x_{i-1}^\ell + b_i^\ell = x_i^\ell + e_{i-1}^\ell \qquad \forall 1 \le i \le |E| \text{ (set } x_0 = e_0 = 0) \tag{7.4}$$

$$b_j^\ell \le \sum_{\ell' \in X(\ell,j)} x_{j-1}^{\ell'} + x_j^{\ell'} \qquad \forall [c,d]_\ell \in \Psi \; \forall E(j) \subset [c,d]_\ell \text{ with } c \notin E(j) \tag{7.5}$$

$$e_j^\ell \le \sum_{\ell' \in Y(\ell,j)} x_j^{\ell'} + x_{j+1}^{\ell'} \qquad \forall [c,d]_\ell \in \Psi \; \forall E(j) \subset [c,d]_\ell \text{ with } d \notin E(j) \tag{7.6}$$

Subject to these constraints we maximize $\sum_{\ell\in L} \sum_{i=1}^{|E|-1} x_i^\ell \cdot w(E(i))$. The intended meaning of the variables is that $x_i^\ell = 1$ if ℓ is active during $E(i)$ and otherwise $x_i^\ell = 0$. Variable $b_i^\ell = 1$ if and only if $E(i)$ is the first atomic segment of an active interval of ℓ, and analogously $e_i^\ell = 1$ if and only if $E(i)$ is the last atomic segment of an active interval of ℓ. Constraints (7.1)–(7.3) immediately ensure Requirements (R1)–(R3), respectively. Constraint (7.4) means that if ℓ is active during $E(i-1)$ ($x_{i-1}^\ell = 1$), then it must either stay active during $E(i)$ ($x_i^\ell = 1$) or the active interval ends with $E(i-1)$ ($e_{i-1}^\ell = 1$), and if ℓ is active during $E(i)$ ($x_i^\ell = 1$) then it must be active during $E(i-1)$ ($x_{i-1}^\ell = 1$) or the active interval begins with $E(i)$ ($b_i^\ell = 1$). Constraint (7.5) enforces that for ℓ to become active with $E(j)$ at least one witness label of $X(\ell,j)$ is active during $E(j-1)$ or $E(j)$. Analogously, Constraint (7.6) enforces that for ℓ to become inactive with $E(j)$ at

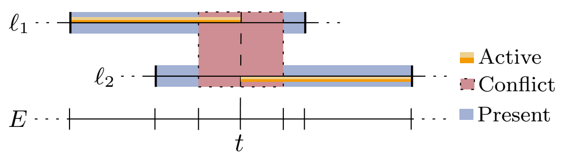


Figure 7.4: The bottom line illustrates possible atomic segments, when assuming that there is a third label that induces the segmentation at time t.

least one witness label of $Y(\ell, j)$ is active during $E(j)$ or $E(j+1)$. Note that without the explicit Constraint (7.5) and Constraint (7.6) two conflicting labels could switch activity at any point during the conflict interval rather than only at the endpoints. For an example see Figure 7.4. The drawing shows an optimal solution that is valid for the ILP formulation if the Constraint (7.5) and Constraint (7.6) are omitted. In particular ℓ_1 becomes inactive at time t, although t is not the right boundary of the corresponding presence interval and there is no conflict of ℓ_1 that begins at t such that the corresponding opponent is active from t on. Analogous observations can be made for ℓ_2. Consequently, this solution does not satisfy AM3.

Theorem 7.1. *Given an instance $I = (L, \Psi, C)$, the ILP (7.1)–(7.6) computes an optimal solution Φ of* GeneralMaxTotal *in AM3. It uses $O(N \cdot (|\Psi| + |C|))$ variables and constraints.*

Proof. Every solution of the ILP corresponds to an activity Φ by defining for every label ℓ the set Φ_ℓ as the set of all maximal intervals in $\bigcup_{i:x_i^\ell=1} E(i)$. Conversely, every valid activity Φ in AM3 can be expressed in terms of the variables of the ILP. To show that we first observe that for every valid activity interval $[a, b]_\ell$ in AM3 the endpoints a and b are necessarily endpoints of a conflict interval or a presence interval of ℓ. Thus $[a, b]_\ell$ can be expressed as the union of consecutive atomic segments represented by the variables x_i^ℓ.

It is clear that the objective function computes the weight of a solution Φ correctly. Thus it remains to show that the Constraints (7.1)–(7.6) indeed model AM3, i.e., every solution of the ILP satisfies AM3 and every activity in AM3 is a solution of the ILP. It follows immediately from the definition of Constraints (7.1)–(7.3) that they model the Requirements (R1)–(R3), assuming that the start- and endpoint of every activity interval is indeed marked by setting $b_i^\ell = 1$ and $e_j^\ell = 1$ for its first and last atomic segments $E(i)$ and $E(j)$. But this is achieved by Constraint (7.4) as discussed above. Now in AM3 a label can only become active (inactive) at the start (end) of its presence interval or at the end (start) of a conflict interval if the conflicting label is active as a witness. We show that Constraint (7.5) yields that the start of an activity interval is correct according to AM3. The argument for the end of an activity interval follows analogously from Constraint (7.6). Let $E(i)$ be the first atomic segment in an activity interval of the label ℓ. Then by Constraint (7.4) we have $b_i^\ell = 1$ and $x_i^\ell = 1$. If $E(i)$ is the first segment of a presence interval then this is a valid start according to AM3.

Note that Constraint (7.5) is not present in that case and thus does not restrict b_i^ℓ. Otherwise let $E(i)$ be not the first segment of a presence interval. Then for this segment the ILP contains Constraint (7.5). If no conflict interval of ℓ ends with $E(i-1)$ then Constraint (7.5) sets $b_i^\ell = 0$ anyways, so this is not possible. If some conflict intervals of ℓ end with $E(i-1)$ but none of them are active in $E(i-1)$ or $E(i)$ then Constraint (7.5) also yields $b_i^\ell = 0$. So the only two possibilities for $b_i^\ell = 1$ are that either $E(i)$ is the first segment of a presence interval or $E(i)$ is the first segment after a conflict interval of ℓ and a witness label active at the beginning of $E(i)$. Thus, every solution of the ILP satisfies AM3.

Conversely, let Φ be valid according to AM3. Since Φ satisfies Requirements (R1)–(R3), the corresponding assignment of binary values to the variables x_i^ℓ, b_i^ℓ, and e_i^ℓ satisfy Constraints (7.1)–(7.4). It remains to show that the Constraints (7.5) and (7.6) hold. Let $[a,b]_\ell \in \Phi$ be a particular activity interval and let $E(i)$ be the atomic segment starting at a. If a is the start of a presence interval of ℓ then there is no Constraint (7.5) for ℓ and the segment $E(i)$ and thus it is possible to have $b_i^\ell = 1$. Otherwise, a is the end of a conflict interval of ℓ with another label ℓ' that is an active witness in the atomic segment $E(i-1)$ or $E(i)$. This means that $x_{i-1}^{\ell'} = 1$ or $x_i^{\ell'} = 1$ and thus Constraint (7.5) is satisfied for $b_i^\ell = 1$. Analogous reasoning for the endpoints of all activity intervals and Constraint (7.6) yield that Φ can indeed be represented as a solution to the ILP.

Since the number of atomic segments is $O(|\Psi| + |C|)$ and there are N labels the bound on the size of the ILP follows. □

We can adapt the above ILP to AM1 and AM2 as follows. For AM2 we replace the right hand side of Constraint (7.5) by 0, and for AM1 we also replace the right hand side of Constraint (7.6) by 0. This excludes exactly the start- and endpoints of the activity intervals that are forbidden in AM1 or AM2. It is easy to see that these ILP formulations can be modified further to solve k-RestrictedMaxTotal by adding the constraint $\sum_{\ell \in L} x_i^\ell \leq k$ for each atomic segment $E(i)$.

Corollary 7.1. *Given an instance $I = (L, \Psi, C)$, GeneralMaxTotal and k-Restricted-MaxTotal can be solved in AM1, AM2, and AM3 by an ILP that uses $O(N \cdot (|\Psi| + |C|))$ variables and constraints.*

Approaches Based on Conflict Graphs. We reduce GeneralMaxTotal to an independent set problem on a weighted conflict graph $G = (V, E)$ such that the maximum weight independent set in G induces the optimal solution of I. Since AM3 is the most general model we first describe the reduction for this variant and then sketch adaptations for AM1 and AM2.

Let $[a,b]_\ell \in \Psi$ be a presence interval of the label $\ell \in L$. If ℓ becomes active within $[a,b]_\ell$, then this happens either at time a or at the end of one of the conflict intervals of $[a,b]_\ell$. Let $s_1, \ldots, s_h$ denote those times. Analogously, if ℓ becomes inactive in $[a,b]_\ell$, then this happens either at time b or at the beginning of one of the conflict intervals of $[a, b]_\ell$. Let $t_1, \ldots, t_h$ denote those times.

Hence, if ℓ is active for an interval $[s, t]_\ell \subseteq [a, b]_\ell$, then there are s_i and t_j with $s_i \leq t_j$ such that $[s, t]_\ell = [s_i, t_j]_\ell$. We call $[s_i, t_j]_\ell$ a *candidate*.

We construct the graph $G_{\text{AM3}} = (V, E)$ as follows. For any presence interval $[a, b]_\ell$ of any label ℓ we introduce a vertex for any candidate $[s_i, t_j]_\ell$ of $[a, b]_\ell$; we identify the vertices with their candidates and assign to each vertex the weight of the candidate. For two candidates u and v of the same presence interval we introduce the edge $\{u, v\}$. Thus, the candidates of the same presence interval form a clique C in G, which we call a *cluster*. For two candidates of different presence intervals $[a, b]_\ell$ and $[c, d]_{\ell'}$ we introduce an edge if and only if $[a, b]_\ell$ and $[c, d]_{\ell'}$ are in conflict during the intersection of both candidates; we say that the corresponding candidates are in conflict.

Conceptually, to construct G_{AM2}, we remove each candidate from G_{AM3} that does not start at the beginning of its presence interval. Further removing each candidate that does not end at the end of its presence interval gives use graph G_{AM1}. Note, however, that in our implementation we constructed G_{AM1} and G_{AM2} directly without G_{AM3}.

Then an independent set $\mathcal{I}$ of G_{AM3} is precisely a set of candidates that are not in conflict. We interpret $\mathcal{I}$ as an activity set of the given instance. We call $\mathcal{I}$ *saturated*, if there are no two candidates $v \in \mathcal{I}$ and $v' \in V \setminus \mathcal{I}$ such that $\mathcal{I}' = \mathcal{I} \cup \{v'\} \setminus \{v\}$ is an independent set, v' and v belong to the same cluster and $w(\mathcal{I}) < w(\mathcal{I}')$, where $w(\mathcal{I}) = \sum_{u \in \mathcal{I}} w(u)$. Note that any maximum weight independent set of G is also saturated.

Lemma 7.1. *Let $\mathcal{I}$ be a saturated independent set of G_{AMX}, then $\mathcal{I}$ is a valid activity set of the instance I with respect to AMX where $X \in \{1, 2, 3\}$.*

Proof. We prove the lemma only for AM3; similar arguments apply for the other two models. Consider the labeling that we obtain by setting the labels' activities according to $\mathcal{I}$. By construction of the candidates, each activity interval in $\mathcal{I}$ is contained in a corresponding presence interval (R1). By construction of the clusters each label is set active at most once for each presence interval (R2). Further, no two labels overlap, because candidates in conflict mutually exclude each other in any independent set of G_{AM3} (R3).

We now prove that $\mathcal{I}$ satisfies AM3 by contradiction. We consider two cases. In the first case there is a label ℓ that is active during a presence interval $[a, b]_\ell$ such that ℓ becomes active at time s with $a < s$ and there is no witness label ℓ' such that a common conflict ends at s. By construction there is an interval $[s, t]_\ell$ in $\mathcal{I}$ for some t. Since $a < s$ there is a further candidate $[s', t]_\ell$ with $s' < s$. Further, we can choose s' such that $[s', t]_\ell$ is not in conflict with any candidate of $\mathcal{I} \setminus \{[s, t]_\ell\}$. Hence, $\mathcal{I}' = \mathcal{I} \cup \{[s', t]_\ell\} \setminus \{[s, t]_\ell\}$ is an independent set of G_{AM3} such that $w(\mathcal{I}') > w(\mathcal{I})$. Consequently, $\mathcal{I}$ is not a saturated independent set, which contradicts the assumption.

In the second case there is a label ℓ that is active during a presence interval $[a, b]_\ell$ such that ℓ becomes inactive at time $t < b$ and there is no witness label ℓ' such that a common conflict begins at t. We can use analogous arguments as in the first case to show that this implied that $\mathcal{I}$ is not saturated. □

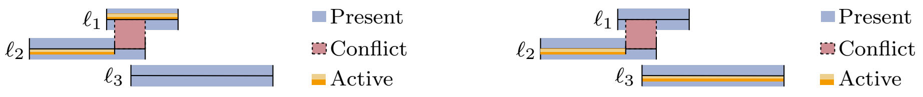
We use different general heuristics for computing independent sets on $G_{\mathrm{AM}X}$ for $X \in \{1, 2, 3\}$. However, those independent sets are not necessarily saturated so that they do not necessarily satisfy the according activity model. Thus, in a post-processing step, we check whether the activity $\mathcal{I}$ satisfies AMX. If this is not the case, then there is a cluster with two vertices $v \in \mathcal{I}$ and $v' \notin \mathcal{I}$ such that $\mathcal{I}' = \mathcal{I} \cup \{v'\} \setminus \{v\}$ is an independent set, and $w(\mathcal{I}) < w(\mathcal{I}')$. We exchange v with v' and repeat the procedure until $\mathcal{I}$ is saturated. We use the following heuristics for computing an independent set $\mathcal{I}$ on $G_{\mathrm{AM}X}$.

Greedy. We first consider GeneralMaxTotal. Starting with an empty solution $\mathcal{I}$, the algorithm removes the candidate c with largest weight from $G_{\mathrm{AM}X}$ and adds it to $\mathcal{I}$. Then, it removes all candidates from $G_{\mathrm{AM}X}$ that are in conflict with c. We repeat this procedure until all candidates are removed from the graph. Since we always take the candidate with largest weight, we can directly conclude that $\mathcal{I}$ is saturated.

In order to solve k-RestrictedMaxTotal for AM1, we create the graph G_{AM1} and apply the procedure as described above. However, this time we remove not only all candidates that are in conflict with the candidate c, but also any candidate that cannot be added to $\mathcal{I}$ without violating the requirement that at most k labels are active at the same time. The resulting activity set $\mathcal{I}$ is then valid with respect to AM1. For AM2 and AM3 we cannot apply the same procedure on G_{AM2} and G_{AM3}, respectively, without potentially violating the requirement of label witnesses; see also Figure 7.5. In our evaluation we therefore use the solutions of AM1 instead, which trivially satisfy AM2 and AM3.

For both GeneralMaxTotal and k-RestrictedMaxTotal the algorithm Greedy is a generalization of the approximation algorithms presented in Chapter 6. The proved approximation guarantees, however, do not apply on Greedy, because we do not make any assumption on the labels' shapes and weights.

PhasedLocalSearch. As a further method to find high-quality solutions for the problem GeneralMaxTotal, we investigated local search algorithms for finding a large-weight independent set in the conflict graph $G_{\mathrm{AM}X}$. As far as we are aware, the Phased Local Search algorithm (PLS) by Pullan [Pul06], originally developed for the maximum (unweighted) clique problem, is the only local search algorithm that has been shown to find maximum or near-maximum independent sets on weighted versions of standard benchmark graphs [Pul09]. Other local search algorithms [JH15] may give higher quality solutions for the unweighted case, but they apply operations that serve only to expand the cardinality of the independent set, which may decrease its weight during the process.



(a) Valid activity for $k = 1$. **(b)** Invalid activity for $k = 1$.

Figure 7.5: Activity of labels for k-RestrictedMaxTotal in AM2 and AM3 for $k = 1$. (a) The optimal solution. (b) A solution wrongly produced on G_{AM2}, G_{AM3}, respectively. For example, Greedy first adds the presence interval of ℓ_3 to the solution $\mathcal{I}$. Then it adds the prefix of ℓ_2's presence interval P that ends at the beginning of the conflict with ℓ_1. It cannot add the whole presence of ℓ_2, because otherwise more than one label is active at the same time. For the same reason it cannot add any part of ℓ_1's presence interval to the solution $\mathcal{I}$. Hence, the end of P is not justified (only ℓ_1 could justify that end). Hence, $\mathcal{I}$ is not valid.

An iteration of PLS consists of repeated *improvements*, which add a vertex to a current independent set $\mathcal{I}$ until it is maximal, followed by a *plateau search*, which swaps a vertex in $\mathcal{I}$ for one that has one neighbor in $\mathcal{I}$. When no improvement or swap can be made, $\mathcal{I}$ is perturbed to include a random vertex. To ensure sufficient diversity of solutions, vertices which are in $\mathcal{I}$ at the end of an iteration are *penalized*, making them less likely to be considered in future iterations. Vertices recover from penalties by a *penalty decrease* mechanism, where penalties are reduced according to a dynamically updated penalty delay parameter. See [Pul06] for further details.

PLS proceeds in three phases, each of which performs iterations using one of three specified vertex selection criteria for choosing an improvement/swap among available candidates, uniformly at random: (1) a *random selection* phase, which selects from all available candidates; (2) a *penalty selection* phase, which selects from candidates with the lowest penalty; and (3) a *greedy selection* phase, which selects from candidates with the lowest degree. The standard PLS algorithm performs 50 iterations of greedy selection, followed by 100 iterations of penalty selection, and 50 iterations of greedy selection, until a stopping criteria is met.

Approach Based on Interval Graphs. The set of presence intervals Ψ induces an *interval graph* H. In this graph the presence intervals form the vertex set and two vertices are connected by an edge if and only if the corresponding intervals intersect. We identify the vertices with the intervals. In particular each vertex has the weight of its presence interval. The next approach makes use of H to compute the activity set Φ.

IntGraph. We first consider GeneralMaxTotal and repeatedly apply the following procedure on H until all vertices are removed from H. We compute a maximum-weight independent set $\mathcal{I}$ on H, which can be done in linear time for interval graphs [HTC92]. We remove those vertices from H and add the intervals to the solution Φ. In case of AM1, we remove also any neighbor of those vertices from H. For AM2, we do not remove those neighbors, but rather shorten the according presence intervals to the longest prefixes that are not in conflict with any presence interval of $\mathcal{I}$. For AM3, we shorten any presence interval of the neighbors to the longest prefix, infix or suffix that is not in conflict with any presence interval of $\mathcal{I}$. Vertices with empty intervals are removed. By design, the activity set Φ is valid according the applied activity model.

When solving k-RestrictedMaxTotal we abort the procedure after the k-th iteration. Since each iteration computes a set of pairwise disjoint intervals, in the computed activity set Φ at most k labels are active at any time.

7.3 Experiments

In this section we present the experimental evaluation of the different models and algorithms for temporal map labeling considering the application of navigation systems. To that end we computed a set of 204 trajectories on the city map of Berlin, which are between 1km and 49km long, with an average length of 20km. We measure the *complexity* of the instances by their input size $|\Psi| + |\Phi|$, which varies between 5 and 10756 and has an average of 1870. We focused on a city map, because the density of the recorded points of interest (POIs) in cities is significantly higher (and thus more challenging) than in the countryside. We obtained the POIs from OpenStreetMap[1] (OSM) data. In order to assess on the usefulness of our approach we modeled the choice of parameters as realistically as possible. However, the setting is an example and can also be specified differently.

7.3.1 Data and Experimental Setup

The trajectories for our experiments were generated from random shortest path queries on the OSM road network of Berlin. Each trajectory is composed of a set of circular arcs and line segments as described in Section 7.2.1. The viewport of the camera is 800 pixels wide and 600 pixels high. Its speed and zoom when moving along the trajectory is determined by the specified speed limit of the underlying road. For each speed limit we introduce a zoom level such that it takes at least 60 seconds for a point to leave at the bottom side of the viewport after entering the viewport on the top side. This improves the legibility of labels moving through the viewport. The change between two zoom levels is done by continuously applying linear interpolation changing the zooming in reasonable time. We took all POIs which are tagged in OSM as *fuel stations, parking lots, ATMs, restaurants, cafés, hotels, motels* and *tourist information* as well as labels for *countries, cities* and *villages* — a set we deemed suitable for car navigation systems. We used the font *Helvetica* in point size 14. We further enforce that any active range of a label lasts at least one seconds to avoid flickering labels. More sophisticated approaches comprising minimum visible area of labels and minimum time between

[1]OpenStreetMap.org

two active phases could be incorporated easily. In this evaluation, however, we focus on the core of our model. Particularly, we weighted the labels equally.

All algorithms were implemented in C++ and compiled with GCC 4.8.3. ILPs were solved by Gurobi 6.0. All experiments were performed on an AMD Opteron 6172 processor clocked at 2.1 GHz, with 256 GB of RAM. Gurobi was allowed to use up to four cores in parallel, while all other experiments were run on a single core. Since we focus on *Phase 2*, we used an easy-to-implement approach for *Phase 1* by sampling the trajectory with high resolution. Much faster, but more laborious approaches can be applied in practice as described in Section 7.2.1.

7.3.2 Evaluation

For each trajectory we ran the different algorithms of Section 7.2.2 for GeneralMaxTotal and k-RestrictedMaxTotal (with $k = 5$ and $k = 10$) in the activity models AM1, AM2, and AM3. Any run exceeding the time limit of 600 seconds was aborted. Similarly when the graph G_{AMX} exceeded 10^7 edges or vertices the run was also aborted to avoid memory overflow. While for IntGraph all runs were processed, the other approaches did not complete all runs. Both Greedy and PhasedLocalSearch completed about 92% of the runs for GeneralMaxTotal, AM3; for the remaining 8%, the graph exceeded the aforementioned size limits. For any other model variant, all runs were completed. Further, Ilp sometimes exceeded its time limit for k-Restricted-MaxTotal, AM2 and AM3: for AM2, Ilp completed about 99% and for AM3 about 66% of the runs. For any other model variant, Ilp completed all its runs.

On each instance, we ran PhasedLocalSearch 10 times and report the average solution size. Each run was made with a different random seed and a time limit of 0.1 seconds. We chose 0.1 seconds, since we observed that PhasedLocalSearch plateaus on nearly all instances after this time. Even with a 100-fold increase to a time limit of 10 seconds, we did not see significant improvement over the solution quality given after 0.1 seconds (see Figure 7.10 in Section 7.5).

In the following, we conduct comparisons between the activity models, the optimization problems GeneralMaxTotal and k-RestrictedMaxTotal as well as comparisons between the applied algorithms.

Activity Models. We compare the activity models AM1, AM2, and AM3 with each other by opposing the optimal solutions obtained by Ilp. Figure 7.6 shows the ratio between the solution for AM2 (AM3) and the solution for AM1 for GeneralMaxTotal. By definition, a solution for AM1 is a lower bound for AM2, which again is a lower bound for AM3. The activity is increased by a factor of 1.06 (1.12) on average for AM2 (AM3). Further, for GeneralMaxTotal the ratio increases with increasing complexity of the instances. Hence, for GeneralMaxTotal the activity models AM2 and AM3

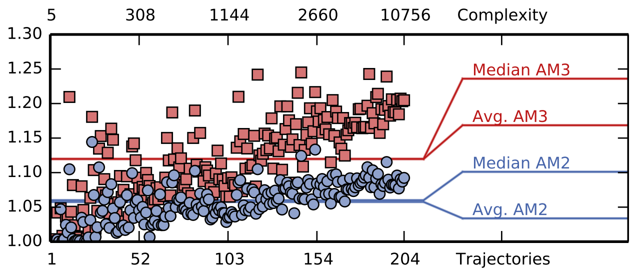


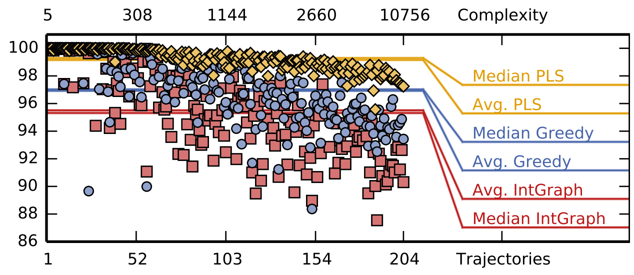
Figure 7.6: GeneralMaxTotal, Comparison of Activity Models. Each data point represents an instance solved by Ilp. X-Axis: Instances are sorted by their complexity ($|\Psi| + |C|$) in increasing order. Y-Axis: Ratio between the optimal solution of AM2 (blue disks) or AM3 (red squares) and the optimal solution for AM1.

increase the amount of displayed information moderately. For general applications such as map exploration this improvement is potentially helpful for the user.

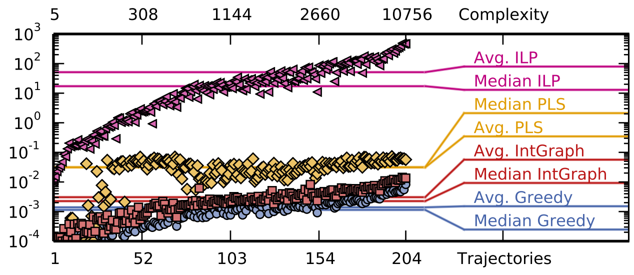
In contrast, for 5-RestrictedMaxTotal the activity is only increased by a factor of 1.02 (1.04) on average for AM2 (AM3); see Figure 7.11 in Section 7.5. For 10-RestrictedMaxTotal we obtain a factor of 1.03 (1.06) on average for AM2 (AM3). For both optimization problems this ratio decreases with the increasing complexity of the instances. Hence, for k-RestrictedMaxTotal the activity models AM2 and AM3 increase the displayed amount of information only slightly, while producing more potentially distracting visual effects by changing the labels' activities during their visibility in the viewport. Keeping in mind that k-RestrictedMaxTotal is targeted for small screen devices such as smartphones and navigation systems, the measured gain of additional information does not necessarily justify the additional visual distractions. Hence, AM2 and AM3 are less relevant in the context of k-RestrictedMaxTotal.

Algorithms for GeneralMaxTotal. Next, we compare the presented algorithms with respect to GeneralMaxTotal and AM1. Figure 7.7(a) shows the activity obtained by single runs in relation to the optimal solution obtained by Ilp. In case that Ilp exceeded the time limit, we used the upper bound that has been found so far by Ilp as reference. If such an upper-bound has not been found by Ilp, the run is omitted in the plot. Figure 7.7(b) shows the running times, again with aborted runs omitted.

Concerning quality PhasedLocalSearch outperforms the two other algorithms. No run achieved less than 95% of the optimal solution, while for Greedy 23% and for IntGraph 45% of the runs achieved less than 95% of the optimal solution. On average PhasedLocalSearch achieved 99% of the optimal solution, while Greedy achieved 97% and PhasedLocalSearch achieved 96% of the optimal solution. Concerning running time PhasedLocalSearch (0.03 sec. in avg.) is slightly slower, then Greedy (0.001 sec.) and IntGraph (0.003 sec.). The running times of Ilp with an average of 51 seconds stayed far behind.



(a) Quality



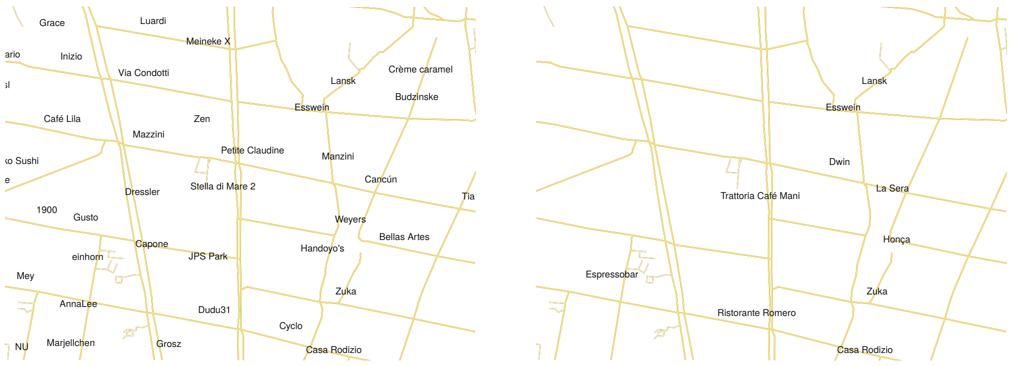
(b) Running Time

Figure 7.7: GeneralMaxTotal, AM1.
Each data point represents an instance solved by IntGraph (red square), Greedy (blue disk) or PhasedLocalSearch (PLS) (yellow diamond). X-Axis: Instances are sorted according their complexity in increasing order. Y-Axis: (a) Achieved percentage of the optimal ILP solution. (b) Running time in seconds (log. scale).

For AM2 and AM3 PhasedLocalSearch is no longer the leader[2] and IntGraph outperforms the other algorithms; see Figure 7.12 in Section 7.5. For AM2 both the average and median (about 89%) stay behind the average and median of Greedy (about 93%) and IntGraph (about 95%). For AM3 this gap is even more pronounced (85% vs. 90% and 94%, respectively). Further, the quality of PhasedLocalSearch is strongly dispersed (minimum 67%). For both activity models AM2 and AM3 Greedy and IntGraph yield similar results concerning quality. However, concerning running time IntGraph clearly beats the other approaches and, unlike the other two approaches, completed every run.

Optimization Models. We now compare GeneralMaxTotal with k-Restricted-MaxTotal. For each trajectory and each integer $n < |L|$ we determined the proportion of the trajectory for which at least n labels are active. For GeneralMaxTotal and AM1 we obtained the following results; similar results hold for AM2 and AM3. On average for over 50% of the trajectory's length more than 3 labels are active at the same time. However, for over 25% (12.5%) of the trajectory's length more than 8 (12) labels are active at the same time, which already may overwhelm untrained observers [Mil56]. Further, for 67% (42%) of the instances there are times when more than 20 (40) labels are active. In some extreme cases over 60 labels are active at the same time. Figure 7.8(a)

[2]The time for PhasedLocalSearch to perform a single iteration depends upon the degree of vertices in the current independent set. Graphs for AM2 and AM3 have much higher vertex degrees than for AM1, which explains why PhasedLocalSearch performs so poorly on these instances.



(a) GeneralMaxTotal **(b)** 10-RestrictedMaxTotal

Figure 7.8: Frame of a dynamic map labeling. While in (a) 54 labels are displayed at the same time, in (b) 10 labels are displayed in order to limit the informational content.

shows a frame of a dynamic map labeling with 54 active labels. We emphasize that the labeling heavily occludes the map such that its remaining content becomes hardly legible. For the application of navigation systems it therefore lends itself to limit the number of simultaneously active labels, which strongly motivates the usefulness of k-RestrictedMaxTotal. Figure 7.8(b) shows the same frame with a labeling produced by Ilp for 10-RestrictedMaxTotal.

Algorithms for k-RestrictedMaxTotal. Finally, we discuss the performance of the algorithms for k-RestrictedMaxTotal with $k = 5$. Similar results hold for the case $k = 10$; see Figure 7.14 in Section 7.5. Recall that PhasedLocalSearch does not support this optimization problem. Figure 7.9 shows the quality ratios and running times for 5-RestrictedMaxTotal in AM1; see also Figure 7.13 in Section 7.5. IntGraph outperforms Greedy both concerning quality and running time. It achieves more than 99% of the optimal solution on average. Further, any run achieves at least 95% of the optimal solution. In contrast, Greedy achieves 96% of the optimal solution on average. Further, 27% of the runs reach less than 95% of the optimal solution, but at least 89%. Further, while IntGraph does not exceed a running time of 0.01 seconds, Greedy needs up to 0.1 seconds. On average, IntGraph took 0.002 seconds and Greedy took 0.01 seconds. The running times of Ilp with an average of 54 seconds stayed far behind.

7.3.3 Discussion

In the above evaluation we considered both the temporal labeling models and several labeling algorithms. From the comparison of the three activity models we conclude that AM1, the most restricted model that does not modify a label's activity during its

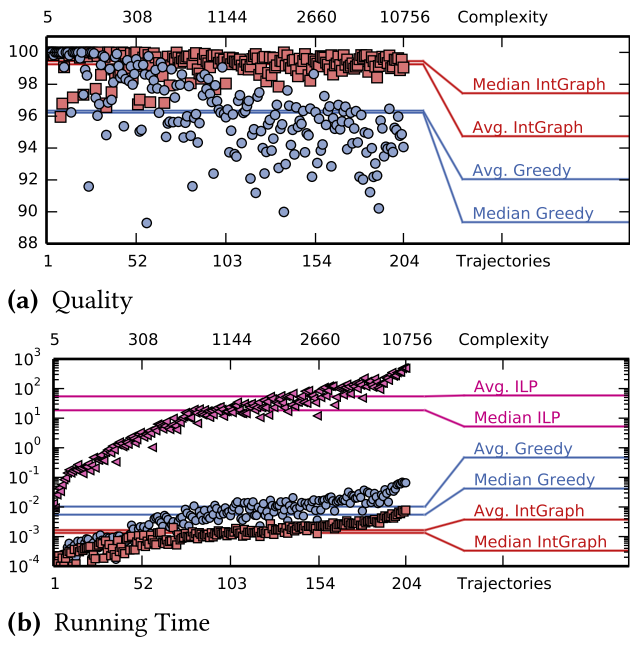


Figure 7.9: 5-RestrictedMaxTotal, AM1. Each data point represents an instance solved by IntGraph (red square) or Greedy (blue disk). X-Axis: Instances are sorted by their complexity ($|\Psi| + |C|$) in increasing order. Y-Axis: (a) Achieved percentage of the optimal ILP solution. (b) Running time in seconds (log. scale).

presence interval and thus fully avoids flickering, is not much worse in terms of the total activity. Further, the quality difference depends on the considered optimization problem: In GeneralMaxTotal the average improvement of AM2 is 6% and of AM3 it is 12%. For k-RestrictedMaxTotal and $k = 10$ the average improvement of AM2 and AM3 is only 3% and 6%, respectively. Whether the gain in displayed content of AM2 and AM3 outweighs the additional flickering effects would need to be examined in a formal user study. The evaluation of the models further showed that without placing any restrictions on the number of simultaneously active labels in GeneralMaxTotal, we frequently observe instances with high numbers of labels, which is not acceptable in certain applications. This justifies the separate consideration of k-RestrictedMaxTotal.

The comparison of the algorithms showed that different algorithms are preferable in different situations. For GeneralMaxTotal and AM1 PhasedLocalSearch outperformed the other algorithms in terms of solution quality. If considering AM2 or AM3 instead or k-RestrictedMaxTotal, we can recommend IntGraph as the best algorithm in both performance measures. Greedy also performs generally well and can be used as an easy-to-implement approach. Ilp provides a simple way to compute optimal solutions and was mainly used to evaluate the other algorithms in terms of solution quality. It could be used directly as a solution approach, but its running time is not reliable and external libraries are needed; thus, we think that the other approaches are preferable in practice.

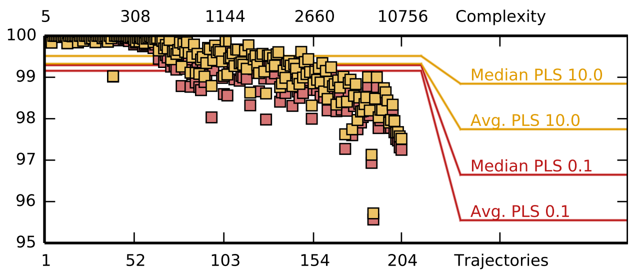
7.4 Conclusions

In a detailed experimental evaluation, we discussed the advantages of different model variants and showed that simple and fast algorithms yield near-optimal solutions for the application of navigation systems.

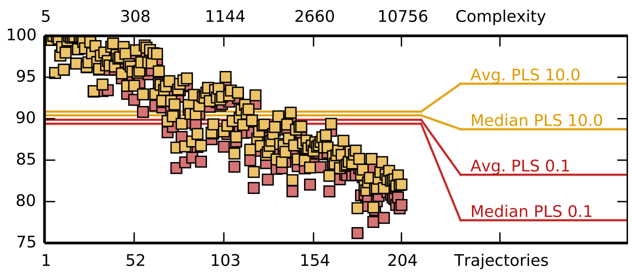
To apply our approach to maps exceeding the size of city maps, we suggest decomposing the conflict graph into smaller components. It seems likely that, when taking countrysides into account, the conflict graph either already consists of several independent components or it contains small cuts that allow for an appropriate decomposition. Further, since our approach essentially relies on algorithms for computing large weighted independent sets in graphs, this is another research direction that promises improvements to our approach.

We focused on the core of our model in order to discuss its application in general. However, with some engineering it can easily be extended to other scenarios or enhanced by further features such as a minimum visible area of labels, different types of map features or labels avoiding obstacles.

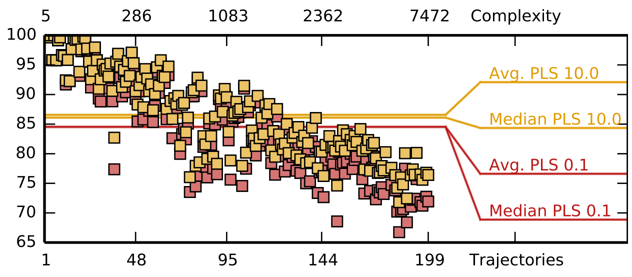
7.5 Additional Plots



(a) AM1

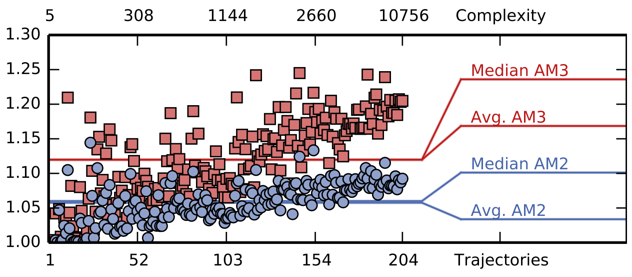


(b) AM2

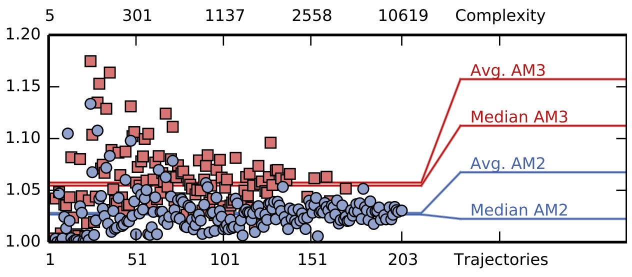


(c) AM3

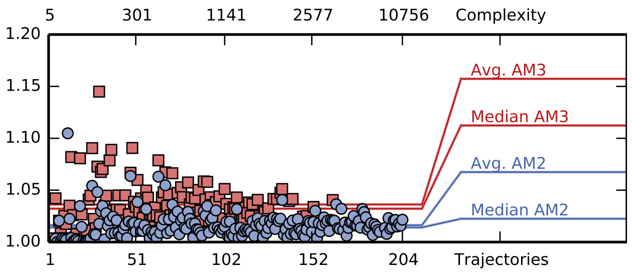
Figure 7.10: GeneralMaxTotal, Quality. Each data point represents an instance solved by PhasedLocalSearch. The local search phase was aborted after 0.1 (red) and 10 (yellow) seconds. X-Axis: Instances are sorted by their complexity ($|\Psi| + |C|$) in increasing order. Y-Axis: Achieved percentage of the optimal ILP solution.



(a) GeneralMaxTotal

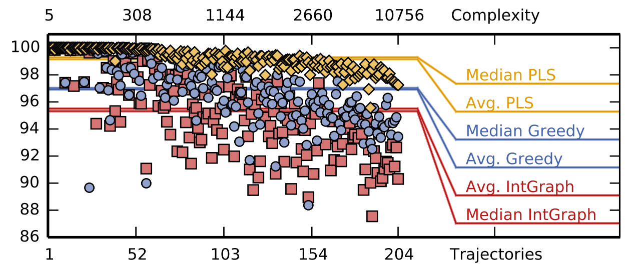


(b) 10-RestrictedMaxTotal

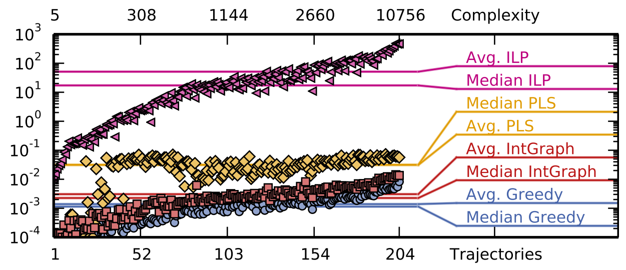


(c) 5-RestrictedMaxTotal

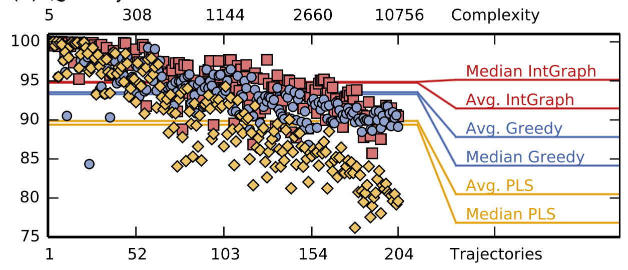
Figure 7.11: Comparison of Activity Models. Each data point represents an instance solved by Ilp. X-Axis: Instances are sorted by their complexity ($|\Psi| + |C|$) in increasing order. Y-Axis: Ratio between the optimal solution of AM2 (AM3) and the optimal solution for AM1.



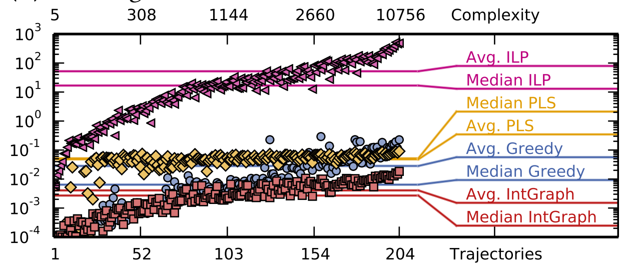
(a) Quality, AM1



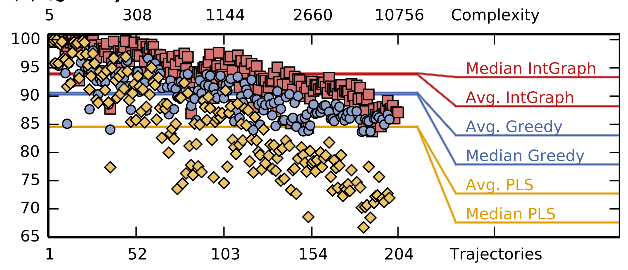
(b) Running Time, AM1



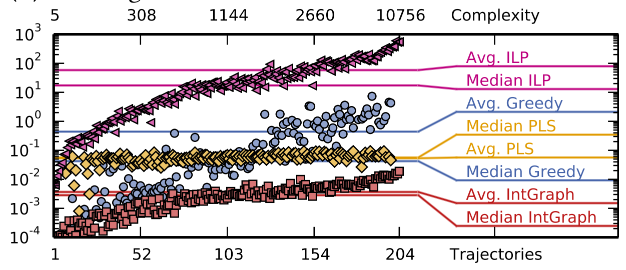
(c) Quality, AM2



(d) Running Time, AM2

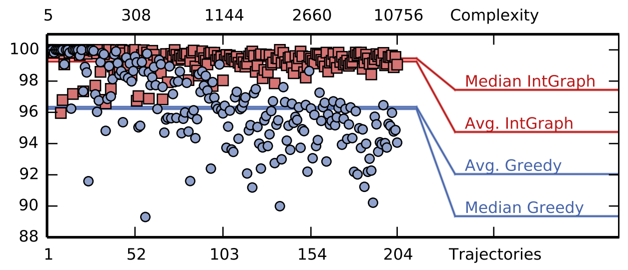


(e) Quality, AM3

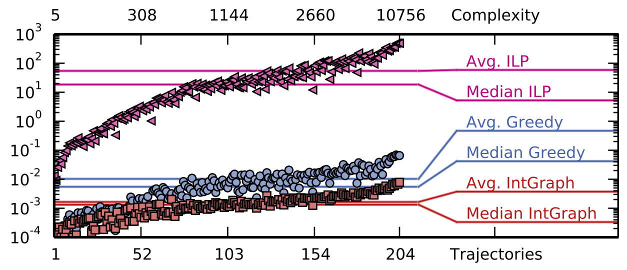


(f) Running Time, AM3

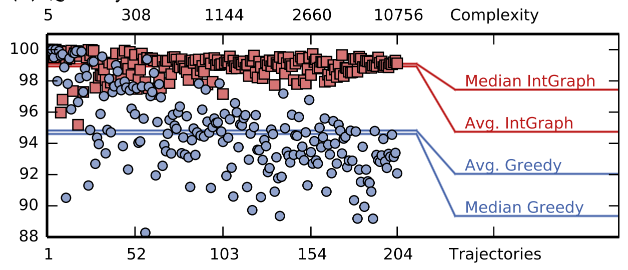
Figure 7.12: GeneralMaxTotal, Quality. Data points represent instances solved by IntGraph (red square) or Greedy (blue disk). Y-Axis: (a),(c),(e): Achieved percentage of the optimal ILP solution. (b),(d),(f): Running time in seconds (log. scale).



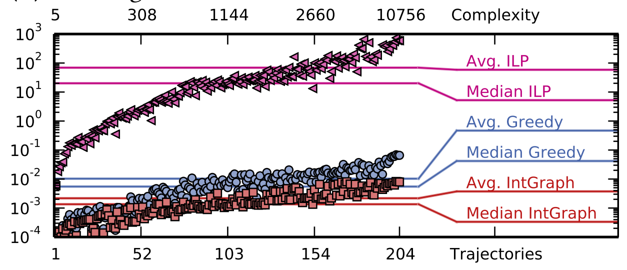
(a) Quality, AM1



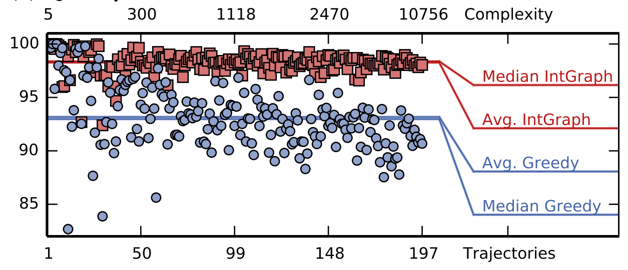
(b) Running Time, AM1



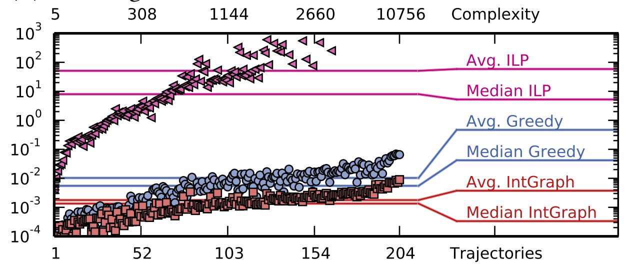
(c) Quality, AM2



(d) Running Time, AM2

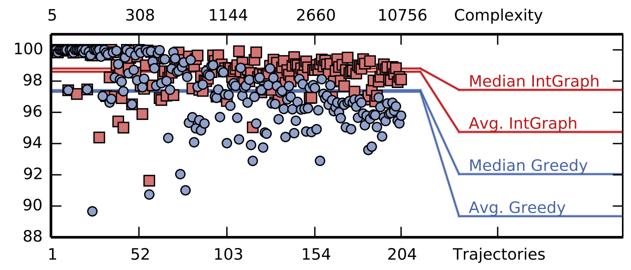


(e) Quality, AM3

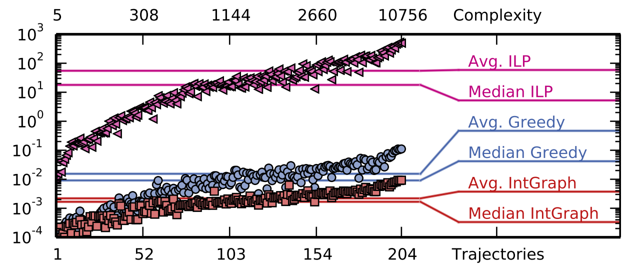


(f) Running Time, AM3

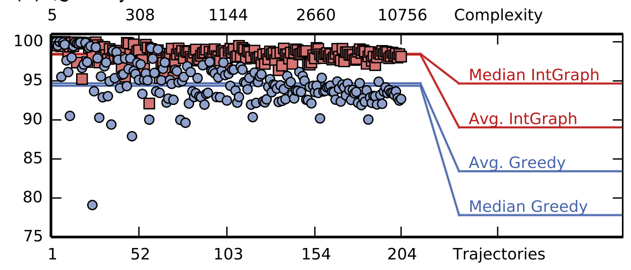
Figure 7.13: 5-RestrictedMaxTotal, Quality. Data points represent instances solved by IntGraph (red square) or Greedy (blue disk). Y-Axis: (a),(c),(e): Achieved percentage of the optimal ILP solution. (b),(d),(f): Running time in seconds (log. scale).



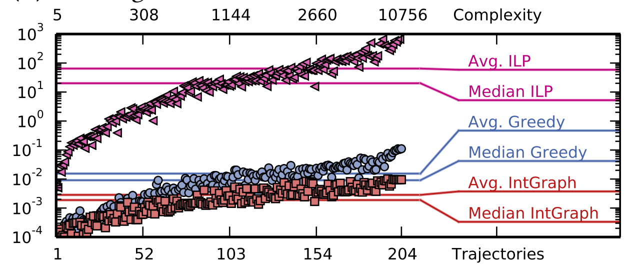
(a) Quality, AM1



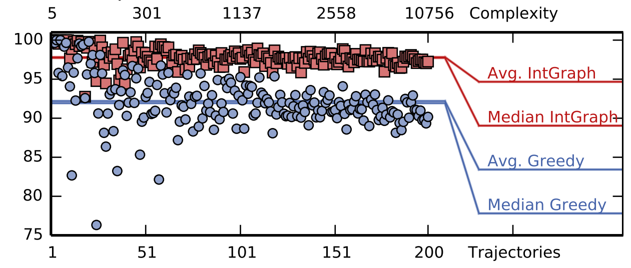
(b) Running Time, AM1



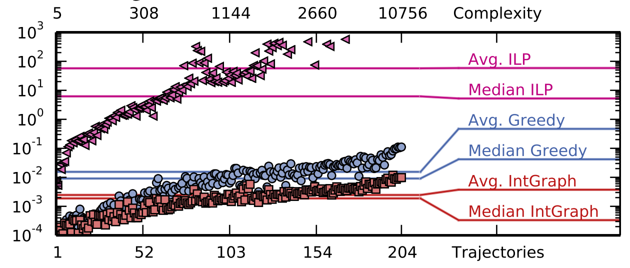
(c) Quality, AM2



(d) Running Time, AM2



(e) Quality, AM3



(f) Running Time, AM3

Figure 7.14: 10-RestrictedMaxTotal, Quality. Data points represent instances solved by IntGraph (red square) or Greedy (blue disk). Y-Axis: (a),(c),(e): Achieved percentage of the optimal ILP solution. (b),(d),(f): Running time in seconds (log. scale).

8 Label Placement in Metro Maps: Model and Theory

Abstract. Drawing network maps automatically comprises two challenging steps, namely laying out the map and placing non-overlapping labels. In this chapter, we tackle the problem of labeling an already existing network map considering the application of metro maps. We present a flexible and versatile labeling model that subsumes different labeling styles. We show that labeling a single line of the network is NP-hard, even if we make very restricting assumptions about the labeling style that is used with this model. For a restricted variant of that model, we then introduce an efficient algorithm that optimally labels a single line with respect to a given cost function. In the subsequent chapter we utilize that algorithm for a generic labeling framework, which we also experimentally evaluate.

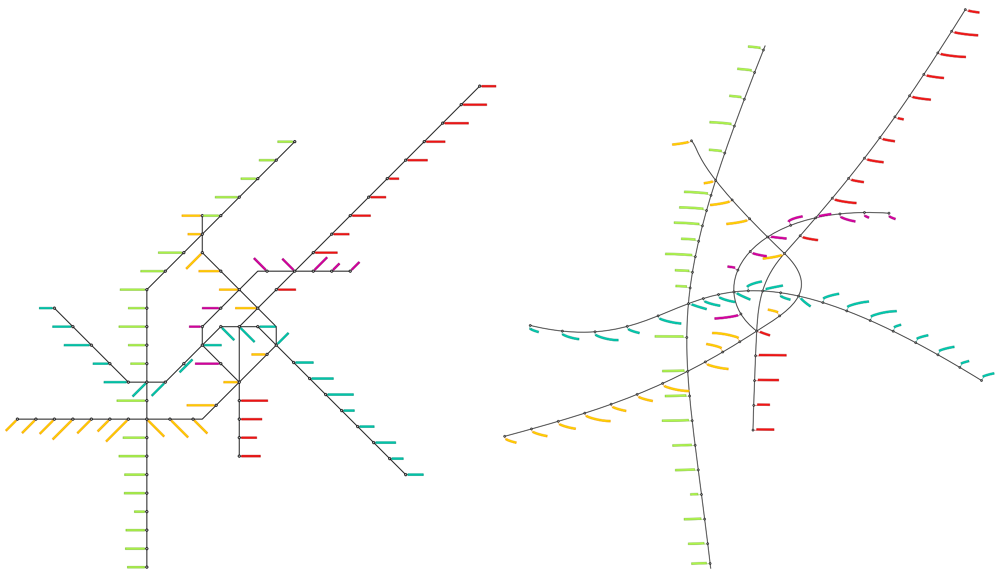
This chapter is based on and partly taken from joint work with Jan-Henrik Haunert [HN15].

8.1 Introduction

Label placement and geographic network visualization are classical problems in cartography, which independently have received the attention of computer scientists. As discussed in Chapter 3, label placement usually deals with annotating map features of interest with text labels such that the associations between the features and the labels are clear and the map is kept legible. Geographic network visualization, on the other hand, often aims at a geometrically distorted representation of reality that allows information about connectivity, travel times, and required navigation actions to be retrieved easily. Computing a good network visualization is thus related to finding a layout of a graph with certain favorable properties [Wol13].

In many applications of network visualization both problems, creating the graph layout and labeling important features, must be solved to the same extent to achieve a usable network map. Take metro maps as an example. To avoid visual clutter in such maps, an *octilinear* graph layout is often chosen, in which the orientation of each edge is a multiple of 45° [NW11, Sto+11, WC11]; see Figure 8.1(a). Alternatively, one may choose a *curvilinear* graph layout, that is, to display the metro lines as curves [Fin+13, Goe+13]; see Figure 8.1(b).

The usefulness of such graph layouts crucially relies on the placement of the stops' names along the metro lines. To guarantee that the user can apply the metro map for its main purpose of navigation, all stops must be labeled and none of the placed



(a) Octilinear graph layout. **(b)** Curved graph layout.

Figure 8.1: Two layouts metro maps of Vienna. Labelings were computed by our algorithm. (a) Octilinear graph layout created by the approach of Nöllenburg and Wolff [NW11]. (b) Curved graph layout created by the approach of Fink et al. [Fin+13].

labels may impair the overall network layout. In this chapter, we investigate the computational problem of placing such labels for the case that the graph layout is already given. We use metro maps as a running example, but our results can also be applied to other kinds of networks maps.

Computing a graph layout for a metro map and labeling the stops have been considered as two different problems that can be solved in succession [WC11], but also integrated solutions have been suggested [NW11, Sto+11]. Nevertheless, in practice, metro maps are often drawn manually by cartographers or designers, as the existing algorithms do not achieve results of sufficient quality in adequate time. For example, Nöllenburg and Wolff [NW11] report that their method needed 10 hours and 31 minutes to compute a labeled metro map of Sydney that they present in their article, while an unlabeled map for the same instance was obtained after 23 minutes—both results were obtained without proof of optimality, but with similar optimality gaps. On the other hand Wang and Chi [WC11] present an algorithm that creates the graph layout and labeling within one second, but they cannot guarantee that labels do not overlap each other or the metro lines.

An integrated approach that simultaneously computes a graph layout and labels all stops allows to take all quality criteria of the final visualization into account. On the other hand, treating both problems separately probably reduces computation time.

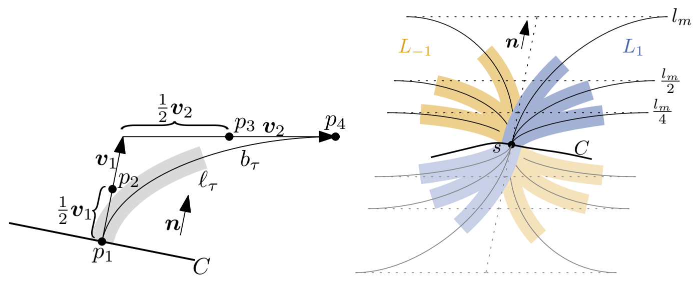
Moreover, we consider the labeling of a metro map as an interesting problem on its own, since, in some situations, the layout of the network is given as part of the input and must not be changed. In a semi-automatic workflow, for example, a cartographer may want to draw or alter a graph layout manually before using an automatic method to place labels, to test multiple different labeling styles with the drawing. Hence, a labeling algorithm is needed that is rather flexible in dealing with different labeling styles for metro maps.

Contribution and Outline. In this chapter, we are given the layout of a metro map consisting of several metro lines on which stops (also called stations) are located. For each stop we are further given its name, which should be placed close to its position. We first introduce a versatile and general model for labeling metro maps; see Section 8.2. Like many labeling algorithms for point sets [AKS98, CMS95, FW91], our algorithm uses a discrete set of candidate labels for each point. Often, each label is represented by a rectangle wrapping the text. Since we also want to use curved labels, however, we represent a label by a simple polygon that approximates a *fat curve*, that is, a curve of certain width reflecting the text height. We then prove that even in that simple model labeling a single metro line is NP-hard considering different labeling styles; see Section 8.3. Hence, we restrict the set of candidates satisfying certain properties, which allows us to solve the problem on one metro line C in $O(n^2)$ time, where n is the number of stops of C; see Section 8.4. In contrast to Bekos et al. [BKS08], who place rectangular labels for stops on a single line segment, we consider arbitrarily shaped labels for stops on general curves. Our algorithm optimizes the labeling with respect to a cost function based on Imhof's [Imh75] classical criteria of cartographic quality; see Section 8.5. The algorithm is not only of theoretical interest, but it provides a very useful subroutine in heuristic algorithms for labeling general metro maps as we demonstrate in Chapter 9.

Note that "stops" on "metro lines" can refer more generally to points of interest on the lines of any kind of a network map. We address labeling styles for octilinear graph layouts and curvilinear graph layouts that use Bézier curves. The more general model behind our method, however, subsumes but is not limited to these particular styles.

8.2 Labeling Model

We assume that the metro lines are given by directed, non-self-intersecting curves in the plane described by polylines, which for example have been derived by approximating Bézier curves. We denote that set of metro lines by $\mathcal{M}$. Further, the stops of each metro line $C \in \mathcal{M}$ are given by an ordered set $\mathcal{S}_C$ of points on C going from the beginning to the end of C. For two stops $s, s' \in \mathcal{S}_C$ we write $s < s'$ if s lies before s'. We denote the union of the stops among all metro lines by $\mathcal{S}$ and call the pair $(\mathcal{M}, \mathcal{S})$ a *metro map*.



(a) Single label. **(b)** Bundle of labels.

Figure 8.2: Construction of curved candidates. (a) Construction of a single label. (b) Candidates $\mathcal{K}_s = L_1 \cup L_{-1}$ for stop s.

For each stop $s \in \mathcal{S}$ we are further given a name that should be placed close to it. In contrast to previous work, we do not follow traditional map labeling abstracting from the given text by bounding boxes. Instead we model a *label* ℓ of a stop $s \in \mathcal{S}$ as a simple polygon. For example, a label could have been derived by approximating a fat curve prescribing the name of the stop; see Figure 8.2. For each stop s we are given a set $\mathcal{K}_s$ of labels, which we also call *candidates* of s. The set $\bigcup_{s \in \mathcal{S}} \mathcal{K}_s$ is denoted by $\mathcal{K}$.

Since "names should disturb other map content as little as possible"[Imh75], we strictly forbid overlaps between labels and lines as well as label-label overlaps. Further, each stop must be labeled. Hence, a set $\mathcal{L} \subseteq \mathcal{K}$ is called a *labeling* if (1) no two labels of $\mathcal{L}$ intersect each other, (2) no label $\ell \in \mathcal{L}$ intersects any metro line $C \in \mathcal{M}$, and (3) for each stop $s \in \mathcal{S}$ there is exactly one label $\ell \in \mathcal{L} \cap \mathcal{K}_s$.

Problem 8.1 (MetroMapLabeling).

Given: *Metro map* $(\mathcal{M}, \mathcal{S})$, *candidates* $\mathcal{K}$ *and cost function* $w\colon 2^{\mathcal{K}} \to \mathbb{R}^+$.
Find: *Optimal labeling* $\mathcal{L}$ *of* $(\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$, *i.e.,* $w(\mathcal{L}) \leq w(\mathcal{L}')$ *for any labeling* $\mathcal{L}' \subseteq \mathcal{K}$, *if a labeling exists.*

The model allows us to create arbitrarily shaped label candidates for a metro map. In our evaluation we have considered two different *labeling styles*. The first style, OctilinStyle, creates for each stop a set of octilinear rectangles as label candidates; see Figure 8.3. We use that style for octilinear maps. The second style, CurvedStyle, creates for each stop a set of fat Bézier curves as label candidates, which are then approximated by simple polygons; see Figure 8.2. We use that style for curvilinear metro maps, in order to adapt the curvilinear style of the metro map. The basic idea is that a label perpendicularly emanates from a stop with respect to its metro line and then becomes horizontal to sustain legibility. In the following section, we motivate our choice of candidates based on cartographic criteria and give detailed technical descriptions for both labeling styles.

8.2.1 Two Examples of Labeling Styles

We extracted the rules for generating label candidates from Imhof's "general principles and requirements" for map labeling [Imh75]. For schematic network maps, the need for "legibility" implies that we must not destroy the underlying design principle with clutter. To this end, we generate candidate labels that adhere to the schematics of the network. That is, we use straight horizontal and diagonal labels with octilinear layouts and curved labels with curvilinear layouts.

We now describe more precisely, how we defined the labeling styles CurvedStyle and OctilinStyle, which are used for curvilinear layouts and octilinear layouts, respectively.

Curvilinear Metro Maps. For CurvedStyle, assume that the given metro map is curvilinear. In order to achieve a "clear graphic association" between a label and the corresponding point p, we construct the simple polygon prescribing a candidate label based on a curve ℓ (possibly a straight-line segment) that emanates from p. The candidate label itself is a continuous section of ℓ that does not directly start in p but at a certain configurable distance from it. We define the end of the candidate label on ℓ based on the text length and assign a non-zero width to the curve section to represent the text height. In the case that p lies on a single curved line C, we require that ℓ and C are perpendicular in p to enhance the angular resolution of the final drawing. By bending ℓ towards the horizontal direction, we avoid steep labels. We approximate ℓ by a simple polygon consisting of a constant number of line segments.

We now describe the construction of a single candidate more specifically. For each stop s of each metro line C we create a constant number of curved labels adapting the curvilinear style of the metro map. The basic idea is that a label perpendicularly emanates from s with respect to C and then becomes horizontal to sustain legibility; see Figure 8.2. Let $\vec{n} = (n_x, n_y)$ be the normalized normal vector of C at s. Further, let $d \in \{-1, 1\}$ and $c_1, c_2 \in \mathbb{R}^+$ be pre-defined constants. For $\tau = (c_1, c_2, d)$ we define the fat cubic Bézier curve b_τ by the following four control points; see Figure 8.2. $p_1 = s$, $p_2 = s + 0.5 \cdot \vec{v}_1$, $p_3 = s + \vec{v}_1 + 0.5 \cdot \vec{v}_2$, $p_4 = s + \vec{v}_1 + \vec{v}_2$, where $\vec{v}_1 = c_1 \cdot \vec{n}$, $\vec{v}_2 = \operatorname{sgn}(\vec{n}) \cdot (d \cdot c_2, 0)$, and $\operatorname{sgn}(\vec{n}) = 1$ if $n_x > 0$ and $\operatorname{sgn}(\vec{n}) = -1$ otherwise. We define the thickness of b_τ to be the pre-defined height of a label. Let ℓ_τ be the sub-curve of b_τ that starts at p_1 and has the length of the name of s and let ℓ'_τ be the curve when mirroring ℓ_τ at s. Further, let l_m be the length of the longest name of a stop in $\mathcal{S}$ and let $L_d = \{\ell_\tau, \ell'_\tau \mid \tau \in \{(l_m, l_m, d), (\frac{l_m}{2}, l_m, d), (\frac{l_m}{4}, l_m, d)\}\}$. If $\vec{n}$ has an orientation less than or equal to 60°, we set $\mathcal{K}_s = L_1$ and otherwise $\mathcal{K}_s = L_1 \cup L_{-1}$. Hence, if $\vec{n}$ is almost vertical and C is therefore almost horizontal at s, we also add the labels L_{-1} pointing into the opposite x-direction than $\vec{n}$. In our experiments we did not let the labels start at s, but with a certain offset to s, in order to avoid intersections with C.

Octilinear Metro Maps. For OctilinStyle assume that the metro map is octilinear. We model the labels as horizontal and diagonal rectangles. Let l be the line segment

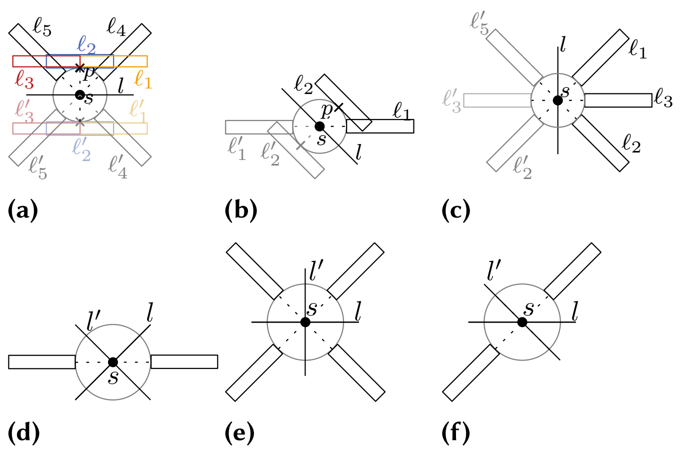


Figure 8.3: Construction of octilinear candidates for a stop s. (a) s lies on a horizontal segment. (b) s lies on a diagonal segment. (c) s lies on a vertical segment. (d) s lies on a crossing of two diagonal segments l and l'. (e) s lies on a crossing of a vertical and horizontal segment. (f) s lies on a crossing of a vertical and diagonal segment.

of C on which s lies and let R be an axis-aligned rectangle that is the bounding box of the name of s. Further, let c be a circle around s with a pre-defined radius. We place the labels such that they touch the boundary of c, but they do not intersect the interior of c. Hence, the labels have a pre-defined offset to s.

If l is horizontal, we place five copies $\ell_1, \dots, \ell_5$ of R above l as follows; see Figure 8.3(a). We place ℓ_1, ℓ_2 and ℓ_3 such that the left-bottom corner of ℓ_1, the midpoint of ℓ_2's bottom edge and the right-bottom corner of ℓ_3 coincides with the topmost point of c. We rotate ℓ_4 by 45° counterclockwise and place it at c such that the midpoint of its left side touches c, i.e., that midpoint lies on a diagonal through s. Finally, ℓ_5 is obtained by mirroring ℓ_4 at the vertical line through s. Mirroring $\ell_1, \dots, \ell_5$ at the horizontal line through s, we obtain the rectangles $\ell'_1, \dots, \ell'_5$, respectively. We then set $\mathcal{K}_s = \{\ell_i, \ell'_i \mid 1 \le i \le 5\}$.

If l is diagonal, we create the candidates in the same manner as in the case that l is horizontal; see Figure 8.3(b). However, we only create the candidates ℓ_2 and ℓ'_2 and the candidates that are horizontally aligned.

If l is vertical, we place three copies $\ell_1, \dots, \ell_3$ of R to the right of l as follows; see Figure 8.3(c). We rotate ℓ_1 by 45° counterclockwise and ℓ_2 by 45° clockwise. We place ℓ_1, ℓ_2, ℓ_3 at c such that the midpoints of their left edges touch c. Mirroring ℓ_1, ℓ_2, ℓ_3 at the vertical line through s, defines the rectangles ℓ'_1, ℓ'_2 and ℓ'_3. We set $\mathcal{K}_s = \{\ell_i, \ell'_i \mid 1 \le i \le 3\}$.

In case that s is a crossing of two metro lines, we create the candidates differently. If s is the crossing of two diagonals, we create the candidates as shown in Figure 8.3(d). If s is the crossing of a horizontal and a vertical segment, we create the labels as shown in Figure 8.3(e). If s is the crossing of a diagonal and horizontal segment, we create the labels as shown in Figure 8.3(f). We analogously create the labels, if s is the crossing of a vertical and a diagonal segment.

Remark: If a stop lies on multiple metro lines, then we can apply similar constructions, where the labels are placed on the angle bisectors of the crossing lines.

8.3 Computational Complexity

We first study the computational complexity of MetroMapLabeling assuming that the labels are either based on OctilinStyle or CurvedStyle. In particular we show that the problem is NP-hard, if the metro map consists of only one line. The proof uses a reduction from the NP-complete problem *monotone planar 3SAT*[Lic82]. Based on the given style, we create for the set C of 3SAT clauses a metro map $(\mathcal{M}, \mathcal{S})$ such that $(\mathcal{M}, \mathcal{S})$ has a labeling if and only if C is satisfiable. The proof can be easily adapted to other labeling styles. Note that the complexity of labeling points using a finite set of axis-aligned rectangular label candidates is a well-studied NP-complete problem, e.g., see [FPT81, FW91]. However, since we do not necessarily use axis-aligned rectangles as labels and since for the considered labeling styles the labels are placed along metro lines, it is not obvious how to reduce a point-feature labeling instance on an instance of MetroMapLabeling. To show the NP-hardness, we prove that it is NP-complete to decide whether a metro map $(\mathcal{M}, \mathcal{S})$ has a labeling based on the given labeling style.

Theorem 8.1. *MetroMapLabeling is NP-hard, if the labels are based on OctilinStyle or CurvedStyle, even if the map has only one metro line.*

Proof. For the illustrations we use OctilinStyle, but the same constructions can be done based on CurvedStyle; see end of proof. We first show that the problem deciding whether $(\mathcal{M}, \mathcal{S})$ has a labeling lies in NP. We first create for each stop $s \in \mathcal{S}$ its candidates $\mathcal{K}_s$ based on the given labeling style. We then guess for each stop $s \in \mathcal{S}$ the label ℓ_s that belongs to the desired labeling $\mathcal{L}$. We can decide in polynomial time whether $\{\ell_s \mid s \in \mathcal{S}\}$ is a labeling of $(\mathcal{M}, \mathcal{S})$ performing basically intersection tests.

We now perform a reduction from the NP-complete Planar Monotone 3-Sat problem [Lic82]. Let φ be a Boolean formula in conjunctive normal form such that it consists of n variables and m clauses and, furthermore, each clause contains at most three literals. The formula φ induces the graph G_φ as follows. G_φ contains for each variable a vertex and it contains for each clause a vertex. Two vertices u and v are connected by an edge $\{u, v\} \in E$ if and only if u represents a variable x and v represents a clause c, such that x is contained in c. We call a clause of φ *positive* (*negative*) if it contains only positive (negative) literals.

The formula φ is an instance of Planar Monotone 3-Sat if

1. φ is monotone, i.e., each clause is either positive or negative, and
2. the graph G_φ is planar and has a rectilinear plane embedding such that
    a) the vertices representing variables are placed on a horizontal line h,
    b) the vertices representing negative clauses are placed below h,

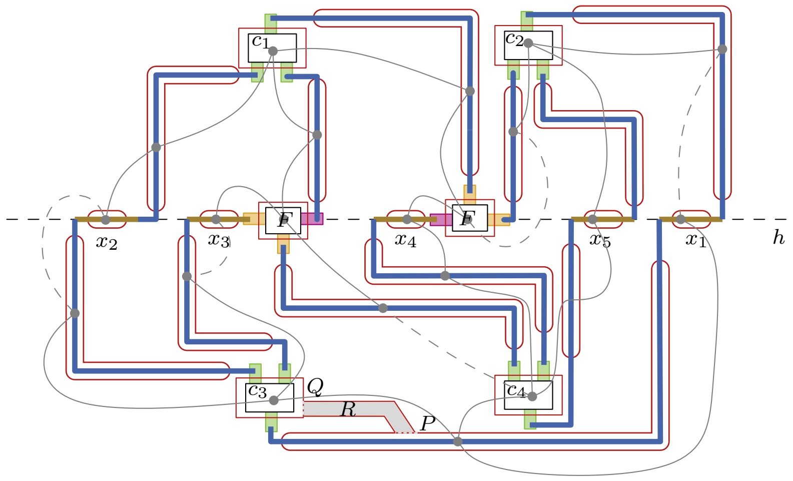


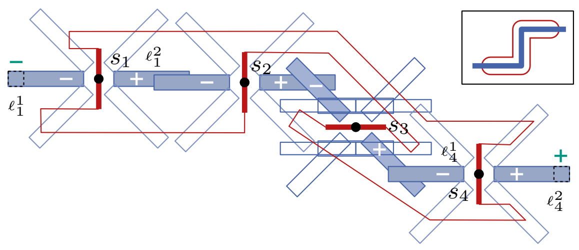
Figure 8.4: Illustration of NP-completeness proof. (a) 3-SAT formula φ with clauses $c_1 = x_4 \vee x_1 \vee x_5$, $c_2 = x_2 \vee x_4 \vee x_3$, $c_3 = \bar{x}_2 \vee \bar{x}_1 \vee \bar{x}_3$ and $c_4 = \bar{x}_3 \vee \bar{x}_5 \vee \bar{x}_4$ represented as a metro map. Truth assignment is $x_1 =$ *true*, $x_2 =$ *true*, $x_3 =$ *false*, $x_4 =$ *false* and $x_5 =$ *false*. The gray graph G represents the adjacencies of the used gadgets. The solid lines represent a spanning tree of G that can be used to merge the polygons to one simple polygon. Exemplarily the polygons P and Q are merged into one polygon R. The single components are illustrated in Figure 8.5.

c) the vertices representing positive clauses are placed above h,
d) the edges are drawn on their respective side of h.

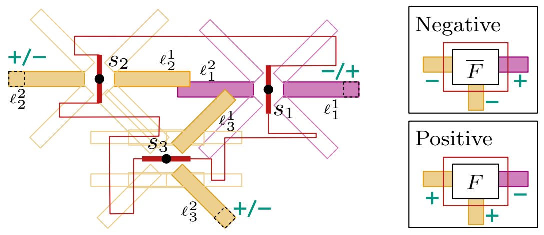
Planar Monotone 3-Sat then asks whether φ is satisfiable.

Using only stops lying on single horizontal and vertical segments, we construct a metro map $(\mathcal{M}, \mathcal{S})$ that mimics the embedding of G_φ. In particular $\mathcal{M}$ will only consist of one metro line C that connects all stops such that the stops and their candidates simulate the variables and clauses of φ. We will prove that $(\mathcal{M}, \mathcal{S})$ has a labeling if and only if φ is satisfiable. We refer to Figure 8.4 for a sketch of the construction. We first define gadgets simulating variables, clauses and connecting structures. Each gadget consists of a set of stops that lie on the boundary of a simple polygon P. Later on we use this polygon P to prescribe the shape of the metro line C.

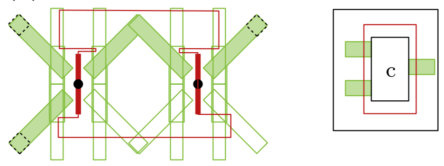
Chain. The *chain gadget* represents and transmits truth values from variables to clauses mimicking the embeddings of the edges in G_φ. A chain consists of an even number of stops $s_1, \ldots, s_k$ that lie on vertical and horizontal segments; see Figure 8.5(a). Hence, with respect to the given labeling style each stop s_i has a predefined set of candidates $\mathcal{K}_{s_i}$. For each stop s_i there are two specially marked candidates ℓ_i^1 and ℓ_i^2 that lie on opposite sides of s_i's segment; for an example see the filled blue labels in Figure 8.5(a). We say that those labels are *selectable*, because we define the gadget such that those labels are the only labels that can be selected for a labeling. To that end, we



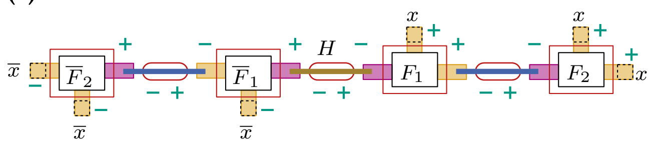
(a) Chain.



(b) Fork.



(c) Clause.



(d) Variable.

Figure 8.5: Illustration of the gadgets. Selectable labels are filled, while all other labels are not filled. Ports are marked with a dashed square. (a) Chain gadget of length 4. (b) Fork gadget. (c) Clause gadget. (d) Variable gadget with three negative and three positive ports.

lay out the metro line such that it does not intersect any selectable label, but all labels that are not selectable. The stops are placed such that the following conditions are satisfied.

(1) The label ℓ_i^2 intersects the label ℓ_{i+1}^1 for $1 \leq i < k$.
(2) Except the intersections mentioned in (1), there is no intersection between selectable labels of different stops.
(3) The segments of the stops are connected by polylines s.t. the result is a simple polygon P intersecting all labels except the selectable labels.

The labels ℓ_1^1 and ℓ_k^2 do not intersect any selectable labels; we call them the *ports* of the chain. Later on, we use the ports to *connect* other gadgets with the chain, i.e., we arrange the gadgets such that two of their ports intersect, but no other selectable label. Further, we assign a polarization to each selectable label. The labels $\ell_1^1, \ldots, \ell_k^1$ are *negative* and the labels $\ell_1^2, \ldots, \ell_k^2$ are *positive*.

Consider a labeling $\mathcal{L}$ of a chain assuming that P is interpreted as a metro line; we can cut P at some point in order to obtain an open curve. By construction of P only selectable labels are contained in $\mathcal{L}$. In particular we observe that if the negative port ℓ_1^1 is not contained in $\mathcal{L}$, then the positive labels $\ell_1^2, \dots, \ell_k^2$ belong to $\mathcal{L}$. Analogously, if the positive port ℓ_k^2 is not contained in $\mathcal{L}$, then the negative labels $\ell_1^1, \dots, \ell_k^1$ belong to $\mathcal{L}$. We use this behavior to represent and transmit truth values through the chain.

Fork. The *fork gadget* splits an incoming chain into two outgoing chains and transmits the truth value represented by the incoming chain into the two outgoing chains. A fork consists of three stops s_1, s_2 and s_3 such that s_1 and s_2 are placed on vertical segments and s_3 is placed on a horizontal segment; see Figure 8.5(b). Analogously to the chain, each stop s_i with $(1 \leq i \leq 3)$ has two *selectable* labels ℓ_i^1 and ℓ_i^2. We arrange the stops such that the following conditions are satisfied.

1. The labels ℓ_2^1 and ℓ_3^1 intersect ℓ_1^2. Apart from those two intersections no selectable label intersects any other selectable label.
2. The segments of the stops are connected by polylines s.t. the result is a simple polygon P intersecting all labels except the selectable labels.

The label ℓ_1^1 is the *incoming port* and the labels ℓ_2^2 and ℓ_3^2 are the *outgoing ports* of the fork. We distinguish two types of forks by assigning different polarizations to the selectable labels. In the *negative* (*positive*) fork, the labels ℓ_1^1, ℓ_2^1 and ℓ_3^1 are *positive* (*negative*) and the labels are ℓ_1^2, ℓ_2^2 and ℓ_3^2 are *negative* (*positive*). Hence, the incoming port is positive (negative) and the outgoings ports are negative (positive).

Consider a labeling $\mathcal{L}$ of a fork assuming that P is interpreted as a metro line. By construction of P only selectable labels belong to $\mathcal{L}$. Further, if the incoming port ℓ_1^1 does not belong to $\mathcal{L}$, then the outgoing ports ℓ_1^2 and ℓ_1^2 belong to $\mathcal{L}$. Finally, if one outgoing port does not belong to $\mathcal{L}$, then the incoming port belongs to $\mathcal{L}$.

Clause. The *clause gadget* represents a clause c of the given instance. It forms a chain of length 2 with the addition that it has three ports instead of two ports; see Figure 8.5(c). To that end one of both stops has three selectable labels; one intersecting a selectable label of the other stop, and two lying on the opposite side of the stop's segment without intersecting any selectable label of the other stop. The gadget is placed at the position where the vertex of c is located in the drawing of G_φ; see Figure 8.4. We observe that a labeling $\mathcal{L}$ of a clause gadget always contains at least one port. Further, we do not assign any polarization to its selectable labels.

Variable. The variable gadget represents a single variable x. It forms a composition of chains and forks that are connected by their ports; see Figure 8.5(d). More precisely, let s be the number of clauses in which the negative literal $\bar{x}$ occurs and let t be the number of clauses in which the positive literal x occurs. Along the horizontal line h on which the vertex of x is placed in the drawing of G_φ, we place a horizontal chain H. Further, we place a sequence of negative forks $\overline{F}_1, \dots, \overline{F}_{s-1}$ to the left of H and a sequence of positive forks $F_1, \dots, F_{t-1}$ to the right of H. The negative incoming port of F_1 is connected to the positive port of H by a chain. Two consecutive forks F_i and F_{i+1} are connected by a chain H' such that H' connects a positive outgoing port of F_i with the negative incoming port of F_{i+1}. Analogously, the positive incoming port of $\overline{F}_1$ is connected to the negative port of H by a chain. Two consecutive forks $\overline{F}_i$ and $\overline{F}_{i+1}$ are connected by a chain H' such that H' connects a negative outgoing port of $\overline{F}_i$ with the positive incoming port of $\overline{F}_{i+1}$.

We observe that the gadget has $s + t$ free ports. Further, we can arrange the forks such that the free ports of $F_1, \ldots, F_{t-1}$ lie above h and the free ports of $\overline{F}_1, \ldots, \overline{F}_{s-1}$ lie below h.

Consider a labeling $\mathcal{L}$ of a variable. By construction of the forks and chains, if one positive free port is not contained in $\mathcal{L}$, then all negative free ports must be contained in $\mathcal{L}$. Analogously, if one negative free port is not contained in $\mathcal{L}$, then all positive free ports must be contained in $\mathcal{L}$.

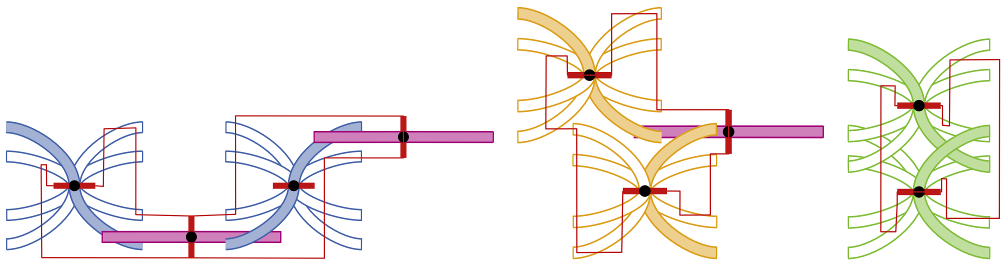
Using additional chains we connect the positive free ports with the positive clauses and the negative ports with the negative clauses correspondingly; see Figure 8.4. More precisely, assume that the variable x is contained in the positive clause c; negative clauses can be handled analogously. With respect to the drawing of G_φ, a positive free port of x's gadget is connected with the negative port of a chain whose positive port is connected with a free port of c's gadget. Note that we can easily choose the simple polygons enclosing the gadgets such that they do not intersect by defining them such that they surround the gadgets tightly.

One Metro Line. We construct the polygons enclosing the single gadgets such that they do not intersect each other. We now sketch how the polygons can be merged to a single simple polygon P. Cutting this polygon at some point we obtain a polyline prescribing the desired metro line.

We construct a graph $H = (V, E)$ as follows. The polygons of the gadgets are the vertices of the graph and an edge (P, Q) is contained in E if and only if the corresponding gadgets of the polygons P and Q are connected by their ports; see Figure 8.4. Since G_φ is planar and the gadgets mimic the embedding of G_φ, it is not hard to see that H is also planar. We construct a spanning tree T of H. If an edge (P, Q) of H is also contained in T, we *merge* P and Q obtaining a new simple polygon R; see for an example Figure 8.4. To that end we cut P and Q in polylines and connect the four end points by two new polylines such that the result is a simple polygon. We in particular ensure that the new polygon does not intersect any other polygon and that R intersects the same labels as P and Q together. In T we correspondingly contract the edge. Note that by contracting edges, T remains a tree. We repeat that procedure until T consists of a single vertex, i.e., only one simple polygon is left.

Soundness. It is not hard to see that our construction is polynomial in the size of the given 3SAT formula φ.

Assume that φ is satisfiable. We show how to construct a labeling $\mathcal{L}$ of the con-



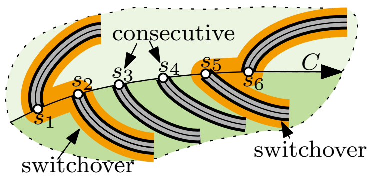
(a) Chain. **(b)** Fork. **(c)** Clause.

Figure 8.6: Illustration of the gadgets based on CurvedStyle. Selectable labels are filled, while all other labels are not filled. (a) Chain gadget of length 4. (b) Fork gadget. (c) Clause gadget.

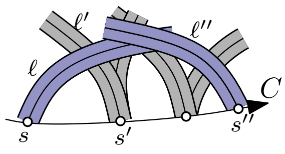
structed metro map. For each variable x that is *true* (*false*) in the given truth assignment, we put all negative (positive) labels of the corresponding variable gadget and its connected chains into $\mathcal{L}$. By construction those labels do not intersect. It remains to select labels for the clause gadgets. Consider a positive clause c; negative clauses can be handled analogously. Since φ is satisfiable, c contains a variable x that is *true* in the given truth assignment of φ. The set $\mathcal{L}$ contains only negative labels of the chain connecting the gadget of x with the gadget of c, but no positive labels of that chain. Hence, we can add the port of c's gadget that is connected to that chain without creating intersections. For the second stop of the clause we put that selectable label into $\mathcal{L}$ that is not a port. We can apply this procedure to all positive and negative clauses without creating intersections, which yields the labeling $\mathcal{L}$ of the constructed metro map.

Finally, assume that we are given a labeling $\mathcal{L}$ of the constructed metro map. Consider the clause gadget of a positive clause c; negative clauses can be handled analogously. By construction $\mathcal{L}$ contains at least one port ℓ of that gadget. This port is connected to a chain, which is then connected to a gadget of a variable x. We set that variable x *true*. We apply this procedure to all clauses; for negative clauses we set the corresponding variable to *false*. Since ℓ is contained in $\mathcal{L}$, only negative labels of that chain can be contained in $\mathcal{L}$, but no positive labels. Hence, the positive ports of the variable gadget of x are also not contained in $\mathcal{L}$. By the previous reasoning this implies that all negative ports of the gadget are contained in $\mathcal{L}$. Consequently, by applying a similar procedure to negative clauses, it cannot happen that x is set to *false*. Altogether, this implies a valid truth assignment of φ.

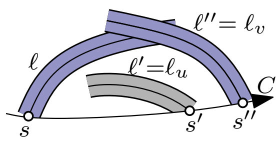
Remarks: Figure 8.6 illustrates the construction of the gadgets for CurvedStyle. Note that only the fork gadget, the clause gadget and the chain gadget rely on the concrete labeling style. Further, using CurvedStyle, a stop s lying on a vertical



(a) Notation.



(b) TP satisfied.



(c) TP not satisfied.

Figure 8.7: (a) Consecutive stops and switchovers. (b) The candidates satisfy the transitivity property (TP). (c) The candidates do not satisfy the transitivity property.

segment l_v has exactly two different distinguish labels; one that lies to the left of l_v and one that lies to the right of l_v. □

8.4 Labeling Algorithm for a Single Metro Line

We now study the case that the given instance $I = (\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$ consists only of one metro line C. Based on cartographic criteria we introduce three additional assumptions on I, which allows us to efficiently solve MetroMapLabeling.

For each stop $s \in S$, we assume that each candidate $\ell \in \mathcal{K}_s$ is assigned to one side of C; either ℓ is a *left candidate* assigned to the left side of C, or ℓ is a *right candidate* assigned to the right side of C. For appropriately defined candidate sets those assignments correspond with the geometric positions of the candidates, i.e., left (right) candidates lie on the left (right) hand side of C.

Assumption 8.1 (Separated Labels)**.** *Candidates that are assigned to different sides of C do not intersect.*

This assumption is normally not a real restriction, because for appropriately defined candidate sets and realistic metro lines, the line C separates both types of candidates geometrically. We further require what we call the *transitivity property.*

Assumption 8.2 (Transitivity Property)**.** *For any three stops $s, s', s'' \in \mathcal{S}$ with $s < s' < s''$ and any three candidates $\ell \in \mathcal{K}_s$, $\ell' \in \mathcal{K}_{s'}$ and $\ell'' \in \mathcal{K}_{s''}$ assigned to the same side of C, it holds that if neither ℓ and ℓ' intersect nor ℓ' and ℓ'' intersect then also ℓ and ℓ'' do not intersect; see also Figure 8.7(b)–(c).*

In Chapter 9 we establish Assumption 8.1 and Assumption 8.2 by removing candidates greedily. We particularly show that for real-world metro maps and the considered candidate sets we remove only few labels, which indicates that those assumptions have only a little influence on the labelings.

Two stops $s, s' \in \mathcal{S}$ with $s < s'$ are *consecutive* if there is no other stop $s'' \in \mathcal{S}$ with $s < s'' < s'$; see Figure 8.7(a). For two consecutive stops $s_1, s_2 \in \mathcal{S}$ we say that each two candidates $\ell_1 \in \mathcal{K}_{s_1}$ and $\ell_2 \in \mathcal{K}_{s_2}$ are *consecutive* and denote the set that contains each pair of consecutive labels in $\mathcal{L} \subseteq \mathcal{K}$ by $P_{\mathcal{L}} \subseteq \mathcal{L} \times \mathcal{L}$. Further, two consecutive labels $\ell_1, \ell_2 \in \mathcal{K}_C$ form a *switchover* (ℓ_1, ℓ_2) if they are assigned to opposite sides of C, where (ℓ_1, ℓ_2) denotes an ordered set indicating the order of the stops of ℓ_1 and ℓ_2. Two switchovers of C are *consecutive* in $\mathcal{L} \subseteq \mathcal{K}$ if there is no switchover in $\mathcal{L}$ in between of both. We define the set of all switchovers in $\mathcal{K}$ by $\mathcal{W}$ and the set of consecutive switchovers in $\mathcal{L} \subseteq \mathcal{K}$ by $\Gamma_{\mathcal{L}} \subseteq \mathcal{W} \times \mathcal{W}$.

Based on cartographic criteria extracted from Imhof's "general principles and requirements" for map labeling [Imh75], we require a cost function $w\colon 2^{\mathcal{K}} \to \mathbb{R}^+$ of the following form; see also Section 8.5 for a detailed motivation of w.

Assumption 8.3 (Linear Cost Function)**.** *For any $\mathcal{L} \subseteq K$ we require*

$$w(\mathcal{L}) = \sum_{\ell \in \mathcal{L}} w_1(\ell) + \sum_{(\ell_1, \ell_2) \in P_{\mathcal{L}}} w_2(\ell_1, \ell_2) + \sum_{(\sigma_1, \sigma_2) \in \Gamma_{\mathcal{L}}} w_3(\sigma_1, \sigma_2),$$

where $w_1\colon \mathcal{L} \to \mathbb{R}$ rates a single label, $w_2\colon P_{\mathcal{L}} \to \mathbb{R}$ rates two consecutive labels and $w_3\colon \Gamma_{\mathcal{L}} \to \mathbb{R}$ rates two consecutive switchovers.

In particular, we define w such that it penalizes the following structures to sustain readability. (1) Steep or highly curved labels. (2) Consecutive labels that lie on different sides of C, or that are shaped differently. (3) Consecutive switchovers that are placed close to each other.

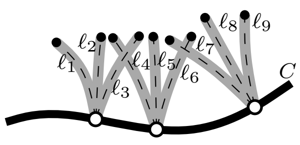
If $I = (\{C\}, \mathcal{S}, \mathcal{K}, w)$ satisfies Assumption 8.1–8.3, we call MetroMapLabeling also SoftMetroLineLabeling. We now introduce an algorithm that solves this problem in $O(n^2k^4)$ time, where $n = |\mathcal{S}|$ and $k = \max\{|\mathcal{K}_s| \mid s \in \mathcal{S}\}$. Note that k is typically constant. We assume w.l.o.g. that $\mathcal{K}$ contains only candidates that do not intersect C.

Labels on One Side. We first assume that all candidate labels in $\mathcal{K}$ are assigned either to the left or to the right side of C; without loss of generality to the left side of C. For two stops $s, s' \in \mathcal{S}$ we denote the instance restricted to the stops $\{s, s'\} \cup \{s'' \in \mathcal{S} \mid s < s'' < s'\}$ by $I[s, s']$. We denote the first stop of C by $\underline{s}$ and the last stop by $\overline{s}$. The transitivity property directly yields the next lemma.

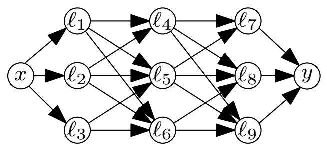
Lemma 8.1. *Let s, s' and s'' be stops with $s < s' < s''$, $\mathcal{L}$ be a labeling of $I[\underline{s}, s']$, $\ell \in \mathcal{L} \cap \mathcal{K}_s$ and $\ell' \in \mathcal{L} \cap \mathcal{K}_{s'}$. Any $\ell'' \in \mathcal{K}_{s''}$ intersecting ℓ also intersects ℓ'.*

Proof. Recall for the proof that we assume that I satisfies Assumptions 8.1–8.3.

Assume for the sake of contradiction that there is a candidate $\ell'' \in \mathcal{K}_{s''}$ such that ℓ'' intersects ℓ but not ℓ'; see Figure 8.7(c). Since $\mathcal{L}$ is a labeling, the labels ℓ and ℓ' do not intersect. Hence, neither ℓ and ℓ' nor ℓ' and ℓ'' intersect. Since all three labels are assigned to the same side of C, the transitivity property holds, which directly contradicts that ℓ and ℓ' do not intersect. □

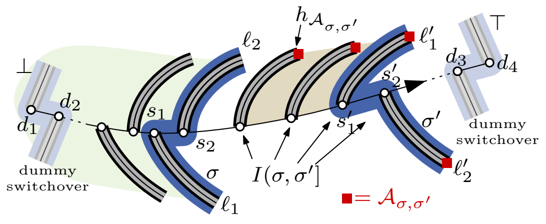


(a) One-sided instance.



$$\begin{tikzcd}
 & \ell_1 \arrow[r] \arrow[dr] \arrow[ddr] & \ell_4 \arrow[r] \arrow[dr] \arrow[ddr] & \ell_7 \arrow[dr] & \\
x \arrow[ur] \arrow[r] \arrow[dr] & \ell_2 \arrow[ur] \arrow[r] \arrow[dr] & \ell_5 \arrow[ur] \arrow[r] \arrow[dr] & \ell_8 \arrow[r] & y \\
 & \ell_3 \arrow[uur] \arrow[ur] \arrow[r] & \ell_6 \arrow[uur] \arrow[ur] \arrow[r] & \ell_9 \arrow[ur] &
\end{tikzcd}$$

(b) Graph G.



(c) Two-sided instance

Figure 8.8: Illustrations for labeling a single metro line. (a) A one-sided instance and (b) the acyclic directed graph G based on its labels. (c) A two-sided instance with a labeling. The switchovers σ' and σ separate the labeling into a two-sided and a one-sided instance.

Hence, the lemma states that ℓ' separates $\mathcal{L}$ from the candidates of the stops succeeding s'. We use this observation as follows. Based on $\mathcal{K}$ we define a directed acyclic graph $G = (V, E)$; see Figure 8.8(a)–(b). This graph contains a vertex u for each candidate $\ell \in \mathcal{K}$ and the two vertices x and y. We call x the *source* and y the *target* of G. Let ℓ_u denote the candidate that belongs to the vertex $u \in V \setminus \{x, y\}$. For each pair $u, v \in V \setminus \{x, y\}$ the graph contains the edge (u, v) if and only if the stop of ℓ_u lies directly before the stop of ℓ_v and, furthermore, ℓ_u and ℓ_v do not intersect. Further, for each vertex u of any candidate of $\underline{s}$ the graph contains the edge (x, u), and for each vertex u of any candidate of $\overline{s}$ the graph contains the edge (u, y). For an edge $(u, v) \in E$ we define its cost w_e as follows. For $u \neq x$ and $v \neq y$ we set $w_e = w_1(\ell_v) + w_2(\ell_u, \ell_v)$. For $x = u$ we set $w_e = w_1(\ell_v)$ and for $v = y$ we set $w_e = 0$.

An x-y *path* $P \subseteq E$ in G is a path in G that starts at x and ends at y. Its costs are $w(P) = \sum_{e \in P} w_e$. The x-y path with minimum costs among all x-y paths is the *shortest* x-y path.

Lemma 8.2. *For any x-y path P in G there is a labeling $\mathcal{L}$ of I with $w(P) = w(\mathcal{L})$ and for any labeling $\mathcal{L}$ of I there is an x-y path P in G with $w(P) = w(\mathcal{L})$.*

Proof. Recall for the proof that we assume that I satisfies Assumptions 8.1–8.3.

Let $P = (V_P, E_P)$ be an x-y path in G and let $\mathcal{L} = \{\ell_v \in \mathcal{K} \mid v \in V_P\}$, where V_P denotes the vertices of P and E_P the edges of P. We show that $\mathcal{L}$ is a labeling of C with $w(\mathcal{L}) = w(P)$. Obviously, for each stop $s \in \mathcal{S}$ the set $\mathcal{L}$ contains exactly one candidate $\ell \in \mathcal{K}_s$. By construction for each edge $(u, v) \in E_P$ the labels ℓ_u and ℓ_v do not intersect. Hence, by Lemma 8.1 the label ℓ_v cannot intersect any label $\ell \in \mathcal{L}$ of any stop that occurs before the stop of ℓ_u. Hence, the set $\mathcal{L}$ is a labeling. Let $\ell_1, \ldots, \ell_n$

be the labels in $\mathcal{L}$ in the order of their stops. It holds

$$w(P) = \sum_{e \in E_P} w_e = w_1(\ell_1) + \sum_{i=2}^{n}(w_1(\ell_i) + w_2(\ell_{i-1}, \ell_i)) = w(\mathcal{L}) \tag{8.1}$$

Now, let $\mathcal{L}$ be an arbitrary labeling of C. We show that there is an x-y path P with $w(P) = w(\mathcal{L})$. Let s and s' be two consecutive stops with $s < s'$ and let ℓ and ℓ' be the corresponding labels in $\mathcal{L}$. Since ℓ and ℓ' do not intersect, the corresponding vertices u and u' of ℓ and ℓ' are adjacent in G. Hence, the labels in $\mathcal{L}$ induce a path P in G. Let $\ell_1, \ldots, \ell_n$ be the labels in $\mathcal{L}$ in the order of their stops. Using Equation (8.1) we obtain $w(P) = w(\mathcal{L})$. □

The lemma in particular proves that a shortest x-y path P in G corresponds with an optimal labeling of I. Due to [Cor+09, Chapter 24], P can be constructed in $O(|V| + |E|)$ time using a dynamic programming approach, which we call MinPath; see also Chapter 2. In particular MinPath considers each edge only once. There are $O(n \cdot k)$ vertices in G and each vertex has at most k incoming edges, which implies that there are $O(n \cdot k^2)$ edges. Since MinPath considers each edge only once, we compute the edges of G on demand, which saves storage.

Theorem 8.2. *If I is one-sided, SoftMetroLineLabeling can be optimally solved in $O(nk^2)$ time and $O(nk)$ space.*

Labels on Both Sides. If candidates lie on both sides of the metro line, we solve the problem utilizing the algorithm for the one-sided case.

Consider a labeling $\mathcal{L}$ of I and let σ, σ' be two switchovers in $\mathcal{L}$ such that σ lies before σ' and no other switchover lies in between both; see Figure 8.8(c). Roughly spoken, σ and σ' induce a two-sided instance that lies before σ and a one-sided instance that lies in between both switchovers σ and σ'.

Lemma 8.3. *Let s, s'_1, s'_2 and s'' be stops with $s < s'_1 < s'_2 < s''$; s'_1 and s'_2 are consecutive. Let $\mathcal{L}$ be a labeling of $I[\underline{s}, s'_2]$, $\ell \in \mathcal{L} \cap \mathcal{K}_s$, $\ell'_1 \in \mathcal{L} \cap \mathcal{K}_{s'_1}$, $\ell'_2 \in \mathcal{L} \cap \mathcal{K}_{s'_2}$ s.t. (ℓ'_1, ℓ'_2) is a switchover. Any $\ell'' \in \mathcal{K}_{s''}$ intersecting ℓ intersects ℓ'_1 or ℓ'_2.*

Proof. Recall for the proof that we assume that I satisfies Assumptions 8.1–8.3.

Assume for the sake of contradiction that there is a label $\ell'' \in \mathcal{K}_{s''}$ such that ℓ'' intersects ℓ without intersecting ℓ'_1 and ℓ'_2. Since ℓ and ℓ'' intersect each other, due to Assumption 8.1 both are assigned to the same side of C; w.l.o.g., let ℓ and ℓ'' be assigned to the left hand side of C. Further, w.l.o.g., let ℓ'_1 be a left candidate and ℓ'_2 a right candidate; analogous arguments hold for the opposite case. Since $\mathcal{L}$ is a labeling, the labels ℓ and ℓ'_1 do not intersect. Hence, neither ℓ and ℓ'_1 nor ℓ'_1 and ℓ'' intersect. Since ℓ, ℓ'_1 and ℓ'' are assigned to the same side of C, the transitivity property must hold. However, this contradicts that ℓ and ℓ'' intersect. □

Hence, the lemma yields that for the one-sided instance in between σ and σ' we can choose any labeling; as long as this labeling does not intersect any label of σ or σ', it composes with σ, σ' and the labeling of the two-sided instance to one labeling for the instance up to σ'. We use that observation as follows.

Let $\sigma = (\ell_1, \ell_2)$ and $\sigma' = (\ell'_1, \ell'_2)$ be two switchovers in $\mathcal{W}$. Let s_1 and s_2 be the stops of ℓ_1 and ℓ_2, and let s'_1 and s'_2 be the stops of ℓ'_1 and ℓ'_2, respectively; see Figure 8.8(c). We assume that $\sigma < \sigma'$, i.e., $s_1 < s'_1$. Let $I(\sigma, \sigma']$ be the instance restricted to the stops $\{s \in \mathcal{S} \mid s_2 < s < s'_1\} \cup \{s'_1, s'_2\}$, where $(\sigma, \sigma']$ indicates that the stops of σ' belong to that instance, while the stops of σ do not.

The switchovers σ and σ' are *compatible* if ℓ_2 and ℓ'_1 are assigned to the same side of C, and there is a labeling for $I[s_1, s'_2]$ such that it contains ℓ_1, ℓ_2, ℓ'_1 and ℓ'_2 and, furthermore, σ and σ' are the only switchovers in that labeling. Let $\mathcal{L}$ be an optimal labeling among those labelings. We denote the labeling $\mathcal{L} \setminus \{\ell_1, \ell_2\}$ of $I(\sigma, \sigma']$ by $\mathcal{A}_{\sigma,\sigma'}$. Utilizing Theorem 8.2, we obtain $\mathcal{A}_{\sigma,\sigma'}$ in $O(n \cdot k^2)$ time.

For any labeling $\mathcal{L}$ of an instance J let $h_{\mathcal{L}} \in \mathcal{L}$ be the label of the first stop in J and let $t_{\mathcal{L}} \in \mathcal{L}$ be the label of the last stop in J; $h_{\mathcal{L}}$ is the *head* and $t_{\mathcal{L}}$ is the *tail* of $\mathcal{L}$. For technical reasons we extend $\mathcal{S}$ by the *dummy stops* d_1, d_2, d_3 and d_4 such that $d_1 < d_2 < s < d_3 < d_4$ for any stop $s \in \mathcal{S}$. For d_1 and d_2 we introduce the *dummy switchover* $\perp$ and for d_3 and d_4 the dummy switchover $\top$. We define that $\perp$ and $\top$ are compatible to all switchovers in $\mathcal{W}$ and that $\perp$ and $\top$ are compatible, if there is a one-sided labeling for I. Conceptually, each dummy switchover consists of two labels that are assigned to both sides of C. Further, neither $\perp$ nor $\top$ has any influence on the cost of a labeling. Hence, w.l.o.g. we assume that they are contained in any labeling.

Similar to the one-sided case we define a directed acyclic graph $G' = (V', E')$. This graph contains a vertex u for each switchover $\mathcal{W} \cup \{\perp, \top\}$. Let σ_u denote the switchover that belongs to the vertex $u \in V$. In particular let x denote the vertex of $\perp$ and y denote the vertex of $\top$. For each pair $u, v \in V$ the graph contains the edge (u, v) if and only if σ_u and σ_v are compatible and $\sigma_u < \sigma_v$. The cost w_e of an edge $e = (u, v)$ in G' is $w_e = w(\mathcal{A}_{\sigma_u,\sigma_v}) + w_3(\sigma_u, \sigma_v) + w_2(\ell^u_2, h_{\mathcal{A}_{\sigma_u,\sigma_v}})$, where $\sigma_u = (\ell^u_1, \ell^u_2)$. In the special case that σ_u and σ_v share a stop, we set $w_e = w_3(\sigma_u, \sigma_v) + w_2(\ell^v_1, \ell^v_2) + w_1(\ell^v_2)$, where $\sigma_v = (\ell^v_1, \ell^v_2)$.

Let P be an x-y path in G' and let $e_1 = (x = v_0, v_1), e_2 = (v_1, v_2), \dots, e_l = (v_{l-1}, v_l = y)$ be the edges of P. For a vertex v_i of P with $0 \leq i \leq l$ we write σ_i instead of σ_{v_i}. We denote the set $\bigcup_{i=1}^{l} \mathcal{A}_{\sigma_{i-1},\sigma_i}$ by $\mathcal{L}_P$.

Lemma 8.4. *a) The graph G' has an x-y path if and only if I has a labeling.*
b) Let P be a shortest x-y path in G', then $\mathcal{L}_P$ is an optimal labeling of I.

Proof. Recall for the proof that we assume that I satisfies Assumption 8.1–8.3.

By construction of G', it directly follows that G' has an x-y path P if and only if I has a labeling $\mathcal{L}$. We first show that $\mathcal{L}_P$ is a labeling of I with $w(\mathcal{L}_P) = w(P)$. Afterwards we prove that for any labeling $\mathcal{L}$ of I it holds $w(\mathcal{L}) \geq w(\mathcal{L}_P)$.

Let $e_1 = (v_0, v_1), e_2 = (v_1, v_2), \ldots, e_l = (v_{l-1}, v_l)$ be the edges of P with $v_0 = x$ and $v_l = y$. For a vertex v_i with $0 \leq i \leq l$ we write σ_i instead of σ_{v_i}. We show by induction over $m \in \mathbb{N}$ with $1 \leq m \leq l$ that

$$\mathcal{L}_m := \bigcup_{i=1}^{m} \mathcal{A}_{\sigma_{i-1},\sigma_i}$$

is a labeling of $I(\sigma_0, \sigma_m]$ with $w(\mathcal{L}_m) = \sum_{i=1}^{m} w_{e_i}$. Altogether this implies that $\mathcal{L}_P = \mathcal{L}_l$ is a labeling with $w(\mathcal{L}_P) = w(P)$.

For $m = 1$ we have $\mathcal{L}_1 = \mathcal{A}_{\sigma_0,\sigma_1}$. By the construction of G' the set $\mathcal{A}_{\sigma_0,\sigma_1}$ is a labeling. Since $\sigma_0 = (\ell_0, \ell_1)$ is a dummy switchover we have $w_3(\sigma_0, \sigma_1) = 0$ and $w_2(\ell_2, h_{\mathcal{A}_{\sigma_0,\sigma_1}}) = 0$. Hence, it holds $w(\mathcal{A}_{\sigma_0,\sigma_1}) = w_{e_1}$.

Now, consider the set $\mathcal{L}_m$ for $m > 1$. We first argue that $\mathcal{L}_m$ is a labeling of $I(\sigma_0, \sigma_m]$. By induction the set $\mathcal{L}_{m-1} \subseteq \mathcal{L}_m$ is a labeling of $I(\sigma_0, \sigma_{m-1}]$. Further, by construction the set $\mathcal{L}_m \setminus \mathcal{L}_{m-1} = A_{\sigma_{m-1},\sigma_m}$ is a labeling of the instance $I(\sigma_{m-1}, \sigma_m]$. Since σ_{m-1} and σ_m are compatible, no label of $\mathcal{L}_m \setminus \mathcal{L}_{m-1}$ intersects any label of σ_{m-1}. Then by Lemma 8.3 no two labels in $\mathcal{L}_m$ intersect each other.

We now show that $w(\mathcal{L}_m) = \sum_{i=1}^{m} w_{e_i}$. By induction we have $w(\mathcal{L}_{m-1}) = \sum_{i=1}^{m-1} w_{e_i}$. Since $\mathcal{L}_m = \mathcal{L}_{m-1} \cup \mathcal{A}_{\sigma_{m-1},\sigma_m}$ it holds

$$w(\mathcal{L}_m) = w(\mathcal{L}_{m-1} \cup \mathcal{A}_{\sigma_{m-1},\sigma_m}). \tag{8.2}$$

We distinguish two cases. First assume that σ_{m-1} and σ_m do not have any stop in common. Let $\sigma_{m-1} = (\ell_1^{m-1}, \ell_2^{m-1})$, then we derive from Equation (8.2)

$$\begin{aligned} w(\mathcal{L}_m) &= w(\mathcal{L}_{m-1} \cup \mathcal{A}_{\sigma_{m-1},\sigma_m}) \\ &= w(\mathcal{L}_{m-1}) + w(\mathcal{A}_{\sigma_{m-1},\sigma_m}) + w_3(\sigma_{m-1}, \sigma_m) + w_2(\ell_2^{m-1}, h_{\mathcal{A}_{\sigma_{m-1},\sigma_m}}) \\ &\overset{(I)}{=} w(\mathcal{L}_{m-1}) + w_{e_m} \overset{(II)}{=} \sum_{i=1}^{m} w_{e_i}. \end{aligned}$$

Equality (I) holds due to the definition of w_{e_m} and Equality (II) is by induction true. Now assume that σ_{m-1} and σ_m have a stop in common. Let $\sigma_m = (\ell_1^m, \ell_2^m)$, then we derive from Equation (8.2)

$$\begin{aligned} w(\mathcal{L}_m) &= w(\mathcal{L}_{m-1} \cup \mathcal{A}_{\sigma_{m-1},\sigma_m}) \\ &= w(\mathcal{L}_{m-1}) + w_3(\sigma_{m-1}, \sigma_m) + w_2(\ell_1^m, \ell_2^m) + w_1(\ell_2^m) \\ &\overset{(III)}{=} w(\mathcal{L}_{m-1}) + w_{e_m} \overset{(IV)}{=} \sum_{i=1}^{m} w_{e_i}. \end{aligned}$$

Equality (III) holds due to the definition of w_{e_m} and Equality (IV) is by induction true. Altogether we obtain that $\mathcal{L}_l$ is a labeling with $w(\mathcal{L}_l) = w(P)$.

Finally we show, that there is no other labeling $\mathcal{L}$ with $w(\mathcal{L}) < w(\mathcal{L}_P)$. Assume for the sake of contradiction that there is such a labeling $\mathcal{L}$. Let $\bot = \sigma_0, \sigma_1, \ldots, \sigma_l = \top$ be the switchovers in $\mathcal{L}$, such that $\sigma_i < \sigma_{i+1}$ for each $0 \leq i < l$. We observe that two consecutive switchovers σ_i and σ_{i+1} are compatible. Hence, for any i with $1 \leq i \leq l$ there is an edge $e_i = (u, v)$ in G' with $\sigma_u = \sigma_{i-1}$ and $\sigma_v = \sigma_{i+1}$. Consequently, the edges $e_1, \ldots, e_l$ form an x-y path Q. By the first two claims of this lemma, there is a labeling $\mathcal{L}_Q$ of I with $w(Q) = w(\mathcal{L}_Q)$. Since P is a shortest x-y path it holds

$$w(\mathcal{L}) < w(\mathcal{L}_P) = w(P) \leq w(Q) = w(\mathcal{L}_Q)$$

We now show that $w(\mathcal{L}_Q) \leq w(\mathcal{L})$ deriving a contradiction. To that end recall that

$$\mathcal{L}_Q = \bigcup_{i=1}^{l} \mathcal{A}_{\sigma_{i-1},\sigma_i}.$$

For $1 \leq i \leq l$ the set $\mathcal{A}_{\sigma_{i-1},\sigma_i}$ is an optimal labeling of the instance $I(\sigma_{i-1}, \sigma_i]$ such that $\mathcal{A}_{\sigma_{i-1},\sigma_i} \cup \sigma_{i-1}$ is a labeling and σ_i is the only switchover contained in $\mathcal{A}_{\sigma_{i-1},\sigma_i}$. Let $\mathcal{L}_i \subseteq \mathcal{L}$ and $\mathcal{L}_i^Q \subseteq \mathcal{L}_Q$ be the labelings restricted to $I(\sigma_{i-1}, \sigma_i]$. In particular we have $\mathcal{L}_i^Q = \mathcal{A}_{\sigma_{i-1},\sigma_i}$ and σ_i is the only switchover in $\mathcal{L}_i$. If we had $w(\mathcal{L}_Q) > w(\mathcal{L})$, there must be two consecutive switchovers σ_{i-1} and σ_i such that $w(\mathcal{L}_i) < w(\mathcal{L}_i^Q)$. However, this contradicts the optimality of $\mathcal{A}_{\sigma_{i-1},\sigma_i}$. Consequently, it holds $w(\mathcal{L}_Q) \leq w(\mathcal{L})$ yielding the claimed contradiction. □

By Lemma 8.4 a shortest x-y path P in G' corresponds with an optimal labeling $\mathcal{L}$ of C, if this exists. Using MinPath we construct P in $O(|V'| + |E'|)$ time. Since $\mathcal{W}$ contains $O(nk^2)$ switchovers, the graph G' contains $O(nk^2)$ vertices and $O(n^2k^4)$ edges. As MinPath considers each edge only once, we compute the edges of G' on demand, which needs $O(nk^2)$ storage. We compute the costs of the incoming edges of a vertex $v \in V'$ utilizing the one-sided case. Proceeding naively, we need $O(nk^2)$ time per edge, which yields $O(n^3k^6)$ time in total.

Reusing already computed information, we improve that result as follows. Let $(u_1, v), \ldots, (u_k, v)$ denote the incoming edges of v such that $\sigma_{u_1} \leq \cdots \leq \sigma_{u_k}$, i.e., the stop of σ_{u_i}'s first label does not lie after the stop of σ_{u_j}'s first label with $i < j$. Further, let $\sigma_v = (\ell_v^1, \ell_v^2)$ and let G_i be the graph for the one-sided instance $I[s_i, s]$ considering only candidates that lie on the same side as ℓ_v^1, where s_i is the stop of the second label of σ_{u_i} and s is the stop of ℓ_v^1. Let P_i be the shortest x_i-y_i path in G_i, where x_i and y_i denote the source and target of G_i, respectively. We observe that excluding the source and target, the graph G_i is a subgraph of G_1 for all $1 \leq i \leq k$. Further, since a subpath of a shortest path is also a shortest path among all paths having the same end vertices, we can assume without loss of optimality that when excluding x_i and y_i from P_i, the path P_i is a sub path of P_1 for all $1 \leq i \leq k$. We therefore only need to compute G_1

and P_1 and can use subpaths of P_1 in order to gain the costs of all incoming edges of v. Hence, we basically apply for each vertex $v \in V$ the algorithm for the one-sided case once using $O(nk^2)$ time per vertex. We then can compute the costs in $O(1)$ time per edge, which yields the next result.

Theorem 8.3. *SoftMetroLineLabeling can be optimally solved in $O(n^2k^4)$ time and $O(nk^2)$ space.*

8.5 Cost Function

In this section we motivate the cost function introduced in Section 8.4. For a given metro map $(\{C\}, \mathcal{S})$ that consists of a single metro line C, and generated candidates $\mathcal{K}$, we rate each labeling $\mathcal{L} \subseteq \mathcal{K}$ using the following cost function:

$$w(\mathcal{L}) = \sum_{\ell \in \mathcal{L}} w_1(\ell) + \sum_{(\ell_1,\ell_2) \in P_{\mathcal{L}}} w_2(\ell_1,\ell_2) + \sum_{(\sigma_1,\sigma_2) \in \Gamma_{\mathcal{L}}} w_3(\sigma_1,\sigma_2),$$

where $w_1 \colon \mathcal{L} \to \mathbb{R}$ rates a single label, $w_2 \colon P_{\mathcal{L}} \to \mathbb{R}$ rates two consecutive labels and $w_3 \colon \Gamma_{\mathcal{L}} \to \mathbb{R}$ rates two consecutive switchovers; see Assumption 8.3. The definition of this function relies on the following considerations, which are based on Imhof's "general principles and requirements" for map labeling [Imh75].

To respect that some (e.g., steep and highly curved) labels are more difficult to read than others, we introduce a cost $w_1(\ell)$ for each candidate label ℓ.

Imhof further notes that "names should assist directly in revealing spatial situation" and exemplifies this principle with maps that show text only while still conveying the most relevant geographic information. To transfer this idea to metro maps, we favor solutions where the labels for each two consecutive stops on a metro line have similar properties. That is, the two labels should be placed on the same side of the line and their slopes and curvatures should be similar. In a map satisfying this criterion, a user need not find the point-text correspondence on a one-to-one basis. Instead, the user can identify metro lines and sequences of stops based on label groups, which, for example, makes it easier to count the stops till a destination. (Of course, this is also an improvement in terms of legibility.) In our model, we consider the similarity of consecutive labels by introducing a cost $w_2(\ell_1, \ell_2)$ for each pair (ℓ_1, ℓ_2) of candidates that belong to consecutive stops on C. We penalize consecutive candidates that lie on opposite sides of the metro line, because those disturb the overall label placement. We add this cost to the objective value of a solution if both candidates are selected. Since we minimize the total costs of the solution, the cost for a pair of candidates should be low if they are similar. Further, if C has labels that do not lie on the same side of C, the implied switchovers should occur in regular distances and not cluttered. Hence, for each pair σ_1 and σ_2 of two consecutive switchovers in a solution, we add a cost $w_3(\sigma_1, \sigma_2)$ to the objective value of the solution that depends on the distance between σ_1 and σ_2; the smaller the distance, the greater the cost of $w_3(\sigma_1, \sigma_2)$.

We now give examples, how w_1, w_2 and w_3 can be specifically defined for CurvedStyle and OctilinStyle. In Chapter 9 we use these concrete choices of w_1, w_2 and w_3 for our evaluation. The definitions depend on the applied labeling style.

Curved Labels. Using CurvedStyle for the labels we define the cost functions as follows. For a label $\ell \in \mathcal{K}$ let $p^1_\ell = (x^1_\ell, y^1_\ell)$ be the start point of ℓ and let $p^2_\ell = (x^2_\ell, y^2_\ell)$ be the end point of ℓ; recall that we derived the labels from Bézier curves. Let $\vec{v}_\ell$ be the vector connecting p^1_ℓ with p^2_ℓ and let $\alpha \in [0, 2\pi]$ denote the angle of $\vec{v}_\ell$. We define

$$\delta_\ell = \begin{cases} \alpha_\ell & 0 \leq \alpha \leq \frac{\pi}{2} \\ \pi - \alpha_\ell & \frac{\pi}{2} < \alpha \leq \pi \\ \alpha_\ell - \pi & \pi < \alpha \leq \frac{3}{2}\pi \\ 2\pi - \alpha_\ell & \frac{3}{2}\pi < \alpha < 2\pi \end{cases}$$

Hence, the angle δ_ℓ is a measure for how horizontal the vector $\vec{v}_\ell$ is, whereby the smaller the value of δ_ℓ, the more horizontal is $\vec{v}_\ell$.

We defined the cost function w_1 rating a single label $\ell \in \mathcal{K}$ to be $w_1 = 10 \cdot \delta_\ell$. Hence, we penalize steep labels. We defined w_2 rating two consecutive labels ℓ_1 and ℓ_2 as follows. If ℓ_1 and ℓ_2 point into different x-direction, w_2 is 150. Else, if ℓ_1 and ℓ_2 are switchovers, the cost w_2 is 0. In all other cases, w_2 is the difference between the angle α_{ℓ_1} and α_{ℓ_2}. Hence, in the latter case we penalize labels that are differently aligned. Finally, w_3 rating two switchovers σ_1 and σ_2 of the same line is defined as $w_3 = \frac{200}{d_{\sigma_1,\sigma_2}}$, where d_{σ_1,σ_2} is the number of stops in between σ_1 and σ_2. In particular w_3 effects that a labeling with equally sized sequences of labels lying on the same side of their metro line are rated better than a labeling where the sequences are sized irregularly.

Octilinear Labels. Using OctilinStyle for the labels we define the cost functions as follows. Recall that we use OctilinStyle for octilinear metro maps. For a label ℓ let l_ℓ be the segment on which its stop is placed. If l_ℓ is horizontal, but ℓ is not diagonal, we set $w_1 = 200$. If l_ℓ is vertical or diagonal, but ℓ is not horizontal, we set $w_1 = 100$. In all other cases we set $w_1 = 0$. The functions w_2 and w_3 are defined in the same way as for CurvedStyle.

8.6 Conclusions

In this chapter, we investigated the problem of labeling metro maps, namely labeling each of its stations with respect to a given objective function. To that end, we followed a candidate-based approach. We showed that selecting an appropriate candidate for each stop is NP-hard in general, even if we only consider a single metro line. Therefore, we relaxed the constraints slightly by introducing some further assumptions on the candidates as well as on the objective function. This allows us to optimally label a single metro line in $O(n^2k^4)$ time. The assumption on the objective function still ensures the possibility to rate single labels, consecutive labels as well as consecutive switchovers, which is crucial to model cartographic criteria appropriately. We further gave a concrete definition of the cost function that is based on cartographic criteria introduced by Imhof [Imh75].

Since metro maps rarely consist of single lines, in the next chapter we present a workflow that utilizes the presented algorithm to label a complete metro map. We particularly show that the assumptions we made have no significant impact on the overall labeling.

9 Label Placement in Metro Maps: An Algorithmic Framework

Abstract. Based on the algorithm for labeling single metro lines introduced in the previous chapter, we present a general and sophisticated workflow for labeling multiple metro lines. Further, we experimentally evaluate our approach on real-world metro maps and show that it achieves near-optimal solutions in appropriate time. To obtain optimal solutions for comparison, we make use of integer linear programming formulations.

This chapter is based on and partly taken from joint work with Jan-Henrik Haunert [HN15].

9.1 Introduction

Metro maps of medium and large cities typically consist of multiple convoluted metro lines: they are connected by transfer stations, run closely in parallel and cross each other. Thus, metro maps often form complex network structures. To draw them in practice, we can utilize the algorithm of the previous chapter that labels a single metro line. Based on that algorithm, we introduce an efficient workflow for labeling metro maps consisting of multiple metro lines; see Section 9.2. Our method is similar to the heuristic presented by Kakoulis and Tollis [KT98], in the sense that it discards some label candidates to establish a set of preconditions that allow for an efficient exact solution. However, our model of quality is more general than the one of Kakoulis and Tollis, because it not only takes the quality of individual labels but also the quality of pairs of labels for consecutive metro stations into account. In Section 9.3, we present alternative approaches that help us to evaluate our approach. This comprises a simple greedy algorithm as well as integer linear programming formulations. In Section 9.4, we present experiments conducted on realistic metro maps. We show that our approach creates high quality labelings in short time. In the remainder of this chapter, we use the notation and definitions introduced in Chapter 8.

9.2 Multiple Metro Lines

In this section we consider the problem that we are given a metro map $(\mathcal{M}, \mathcal{S})$ consisting of multiple metro lines. We present an algorithm that creates a labeling for $(\mathcal{M}, \mathcal{S})$ in two phases. Each phase is divided into two steps; see Figure 9.1 for a schematic illustration. In the first phase the algorithm creates the set $\mathcal{K}$ of label candidates and

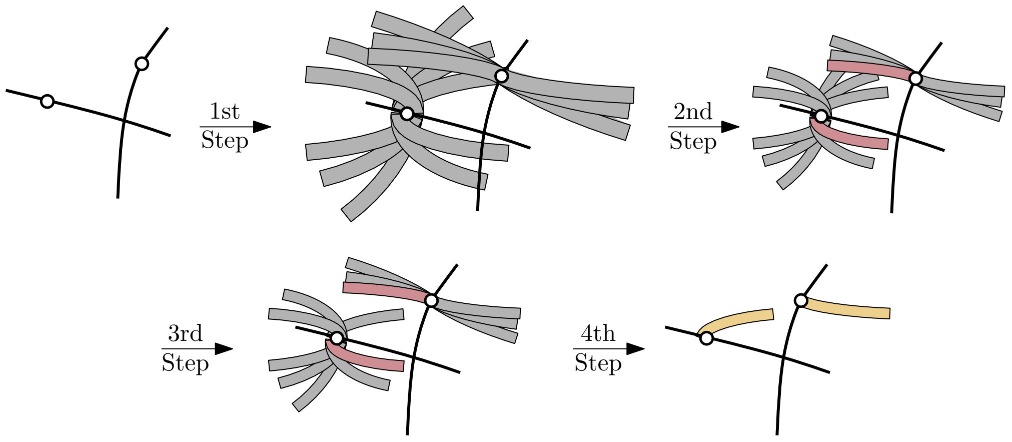


Figure 9.1: Schematic illustration of the presented workflow. 1st Step: generation of candidates. 2nd Step: scaling and creation of initial labeling (red labels). 3rd Step: pre-selection of candidates. 4th Step: Solving the metro lines independently.

ensures that there exists at least one labeling for the metro map. In the second phase it then computes a labeling $\mathcal{L}$ for $(\mathcal{M}, \mathcal{S}, \mathcal{K})$. To that end it makes use of the labeling algorithm for a single metro line; see Section 8.4. In order to rate $\mathcal{L}$, we extend the cost function w for a single metro line on multiple metro lines, i.e., for any $\mathcal{L} \subseteq \mathcal{K}$ we require

$$w(\mathcal{L}) = \sum_{C \in \mathcal{M}} w(C) \text{ with } w(C) = \sum_{\ell \in \mathcal{L}_C} w_1(\ell) + \sum_{(\ell_1, \ell_2) \in P_{\mathcal{L}_C}} w_2(\ell_1, \ell_2) + \sum_{(\sigma_1, \sigma_2) \in \Gamma_{\mathcal{L}_C}} w_3(\sigma_1, \sigma_2),$$

where $\mathcal{L}_C = \mathcal{L} \cap \mathcal{K}_C$ is the labeling restricted to metro line $C \in \mathcal{M}$, $w_1\colon \mathcal{L}_C \to \mathbb{R}$ rates a single label, $w_2\colon P_{\mathcal{L}_C} \to \mathbb{R}$ rates two consecutive labels and $w_3\colon \Gamma_{\mathcal{L}_C} \to \mathbb{R}$ rates two consecutive switchovers of C. In particular w satisfies Assumption 8.3 for a single metro line.

Altogether, the workflow yields a heuristic that relies on the conjecture that using optimal algorithms in single steps is sufficient to obtain good labelings. In our evaluation we call that approach DpAlg.

9.2.1 First Phase – Candidate Generation

First, we create the label candidates $\mathcal{K}$. We then enforce that there is a labeling $\mathcal{L}$ for the given instance $I = (\mathcal{M}, \mathcal{S}, \mathcal{K})$.

1st Step – Candidate Creation. Depending on the labeling style, we generate a discrete set of candidate labels for every stop. Hence, we are now given the instance $(\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$. In particular we assume that each candidate $\ell \in \mathcal{K}$ is assigned to one side of its metro line C, namely to the left or right side of C, and, furthermore, w is a cost function satisfying Assumption 8.3 for each metro line C.

2nd Step – Scaling. Since each stop of a metro map must be labeled, we first apply a transformation on the given candidates to ensure that there is at least one labeling of the metro map. To that end we first determine for each stop $s \in \mathcal{S}$ of each metro line $C \in \mathcal{M}$ two candidates of $\mathcal{K}_s$ that are assigned to opposite sides of C. More specifically, among all candidates in $\mathcal{K}_s$ that are assigned to the right hand side of C and that do not intersect any metro line of $\mathcal{M}$, we take that candidate $\ell_{\mathrm{R}} \in \mathcal{K}_s$ with minimum costs, i.e., $w_1(\ell)$ is minimal. If such a candidate does not exist, because each label of $\mathcal{K}_s$ intersects at least one metro line, we take a pre-defined label $\ell_{\mathrm{R}} \in \mathcal{K}_s$ that is assigned to the right hand side of C. In the same manner we choose a candidate $\ell_{\mathrm{L}} \in \mathcal{K}_s$ that is assigned to the left hand side of C. Let $D_s = \{\ell_{\mathrm{R}}, \ell_{\mathrm{L}}\}$, we now enforce that there is a labeling $\mathcal{L}$ of I such that for each stop $s \in \mathcal{S}$ it contains a label of D_s.

We check whether the set $\bigcup_{s \in \mathcal{S}} D_s$ admits a labeling $\mathcal{L}$ for I. Later, we describe more specifically how to do this. If $\mathcal{L}$ exists, we continue with the third step of the algorithm using I and $\mathcal{L}$ as input. Otherwise, we scale all candidates of $\mathcal{K}$ smaller by a constant factor and repeat the described procedure. Sampling a pre-defined scaling range $[x_{\min}, x_{\max}]$, we find in that manner a scaling factor $x \in [x_{\min}, x_{\max}]$ for the candidates that admits a labeling $\mathcal{L}$ of I. We choose x as large as possible. If we could not find x, e.g, because we have chosen $[x_{\min}, x_{\max}]$ or the sampling not appropriately, we abort the algorithm, stating that the algorithm could not find a labeling.

Next, we describe how to check whether there is a labeling for $\bigcup_{s \in \mathcal{S}} D_s$. Since for each $s \in \mathcal{S}$ the set D_s contains two candidates, we can make use of a 2SAT formulation to model the labeling problem. For each stop $s \in \mathcal{S}$ and each candidate $\ell \in D_s$, we introduce the Boolean variable x_ℓ. Those Boolean variables induce the set $\mathcal{L} = \{\ell \in \bigcup_{s \in \mathcal{S}} D_s \mid x_\ell \text{ is true}\}$. The following formulas are satisfiable if and only if $\mathcal{L}$ is a labeling of $\mathcal{M}$.

$$\begin{aligned} \neg x_\ell \quad & \forall s \in \mathcal{S}\ \forall \ell \in D_s \text{ s.t. } \exists C \in \mathcal{M} \text{ that intersects } \ell \\ \neg x_\ell \vee \neg x_{\ell'} \quad & \forall \ell, \ell' \in \bigcup_{s \in S} D_s \text{ s.t. } \ell \text{ and } \ell' \text{ intersect.} \\ x_\ell \vee x_{\ell'}, \neg x_\ell \vee \neg x_{\ell'} \quad & \forall s \in \mathcal{S}\ \forall \ell, \ell' \in D_s \end{aligned}$$

The first formula ensures that there is no label of the solution that intersects any metro line. The second one avoids overlaps between labels, while the two last formulas enforce that for each stop $s \in \mathcal{S}$ there is exactly one label of D_s that is contained in the solution.

According to [APT79] in linear time with respect to the number of variables and formulas, the satisfiability can be checked. We introduce $O(n)$ variables and instantiate the second formula $O(n^2)$ times, because each pair of candidates may overlap. The remaining formulas are instantiated in $O(n^2)$ time. Hence, the total running time is in $O(n^2 \cdot t)$, where t denotes the number of scaling steps.

9.2.2 Second Phase – Candidate Selection

We assume that we are given the scaled instance $I = (\mathcal{M}, \mathcal{S}, \mathcal{K})$ and the labeling $\mathcal{L}$ for I of the previous phase. We apply a pre-selection on the candidates by discarding candidates such that no two stops' candidates of different metro lines intersect and each metro line satisfies Assumption 8.1 and Assumption 8.2. We never remove a label in $\mathcal{L}$ from the candidate set, however, to ensure that there is always a feasible solution. Finally, considering the metro lines independently, we select for each stop a candidate to be its label using the dynamic program described in Section 8.4.

3th Step – Candidate Pre-Selection. We first ensure that I satisfies Assumption 8.1 and Assumption 8.2. If two candidates ℓ and ℓ' of the same metro line C intersect and if they are assigned to opposite sides of C, we delete one of both labels as follows. If $\ell \in \mathcal{L}$, we delete ℓ', and if $\ell' \in \mathcal{L}$ we delete ℓ. Otherwise, if none of both is contained in $\mathcal{L}$, we delete that label with higher costs; ties are broken arbitrarily. Afterwards, Assumption 8.1 is satisfied. Further, for each metro line C of I we iterate through the stops of C from its beginning to its end. Doing so, we delete candidates from $\mathcal{K}$ violating Assumption 8.2 as described as follows. Let s be the currently considered stop. For each candidate $\ell \in \mathcal{K}_s$ we check for each stop s' with $s' < s$ whether there is a label $\ell' \in \mathcal{K}_{s'}$ that intersects ℓ. If ℓ' exists, we check whether each candidate $\ell'' \in \mathcal{K}_{s''}$ of any stop s'' with $s' < s'' < s$ intersects ℓ or ℓ'. If this is not the case, we delete ℓ if ℓ is not contained in $\mathcal{L}$, and otherwise we delete ℓ'. Note that not both can be contained in $\mathcal{L}$. By construction each metro line of instance I then satisfies Assumption 8.2, where $\mathcal{L} \subseteq \mathcal{K}$.

Finally, we ensure that the metro lines in $\mathcal{M}$ become *independent* in the sense that no candidates of stops belonging to different metro lines intersect and no candidate intersects any metro line. Hence, after this step, the metro lines can independently be labeled such that the resulting labelings compose to a labeling of I.

We first rank the candidates of $\mathcal{K}$ as follows. For each metro line $C \in \mathcal{M}$ we construct a labeling $\mathcal{L}_C$ using the dynamic program for the two sided case as presented in Section 8.4. Due to the previous step, those labelings exist. Note that for two metro lines $C, C' \in \mathcal{M}$ there may be labels $\ell \in \mathcal{L}_C$ and $\ell' \in \mathcal{L}_{C'}$ that intersect each other. For each candidate $\ell \in \mathcal{K}$ we set $val_\ell = 1$ if $\ell \in \mathcal{L}_C$ for a metro line $C \in \mathcal{M}$, and $val_\ell = 0$ otherwise. A candidate $\ell \in \mathcal{K}$ has a *smaller rank* than a candidate $\ell' \in \mathcal{K}$, if $val_\ell > val_{\ell'}$ or $val_\ell = val_{\ell'}$ and $w_1(\ell) \leq w_1(\ell')$; ties are broken arbitrarily.

We greedily remove candidates from $\mathcal{K}$ until all metro lines are independent as follows. We create a conflict graph $G = (V, E)$ such that the vertices of G are the candidates and the edges model intersections between candidates, i.e., two vertices are adjacent if and only if the corresponding labels intersect. Then, we delete all vertices whose corresponding labels intersect any metro line. Afterwards, starting with $\mathcal{I} = \emptyset$, we construct an independent set $\mathcal{I}$ on G as follows. First, we add all vertices of G to $\mathcal{I}$, whose labels are contained in $\mathcal{L}$, and delete them and their neighbors from G. Since the labels in $\mathcal{L}$ do not intersect, $\mathcal{I}$ is an independent set of the original conflict graph. In the increasing order of their ranks, we remove each vertex v and its neighbors from G. Each time we add v to $\mathcal{I}$; obviously sustaining that $\mathcal{I}$ is an independent set in G. We then update for each stop $s \in \mathcal{S}$ its candidate set $\mathcal{K}_s$ to $\mathcal{K}_s \leftarrow \{\ell \in \mathcal{K}_s \mid \text{vertex of } \ell \text{ is contained in } \mathcal{I}\}$. Since all labels of $\mathcal{L}$ are contained in $\mathcal{I}$, there is a labeling for $(\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$ based on the new candidate set $\mathcal{K}$.

4th Step – Final Candidate Selection. Let $I = (\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$ be the instance after applying the third step. By the previous step the metro lines are in the sense independent that candidates of stops belonging to different metro lines do not intersect. Further, they all satisfy Assumption 8.1 and Assumption 8.2. Hence, we use the dynamic programming approach of Section 8.4 in order to label them independently. The composition of those labelings is then a labeling $\mathcal{L}$ of I.

9.3 Alternative Approaches

We now present the three approaches IlpAlg, ScaleAlg, GreedyAlg, which are adaptions of our workflow. We use these to experimentally evaluate our approach against alternatives. While GreedyAlg is a simple and fast greedy algorithm, IlpAlg and ScaleAlg are based on integer linear programming formulations.

9.3.1 Integer Linear Programming Formulation

To assess the impact of the second phase of our approach, we present an integer linear programming formulation that optimally solves MetroMapLabeling with respect to the required cost function. Let $(\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$ be an instance of that problem, which we obtain after the first phase of our approach. We first note that we apply the specific cost function presented in Section 8.5. The cost function w_3, which rates two consecutive switchovers $\sigma = (\ell_1, \ell_2)$ and $\sigma' = (\ell_1', \ell_2')$ of a labeling $\mathcal{L}$, does not rely on the actual switchovers, but only on their positions on the corresponding metro line. Hence, we may assume that w_3 expects the stops of ℓ_1 and ℓ_1'. This assumption helps us reduce the number of variables.

For each $\ell \in \mathcal{K}$ we define a binary variable $x_\ell \in \{0, 1\}$. If $x_\ell = 1$, we interpret it such that ℓ is selected for the labeling. We introduce the following constraints.

$$x_\ell = 0 \qquad \forall \ell \in \mathcal{K} \text{ that intersect a metro line.} \tag{9.1}$$

$$x_\ell + x_{\ell'} \leq 1 \qquad \forall \ell, \ell' \in \mathcal{K} \text{ that intersect.} \tag{9.2}$$

$$\sum_{\ell \in \mathcal{K}_s} x_\ell = 1 \qquad \forall s \in S \tag{9.3}$$

Moreover, for each metro line $C \in \mathcal{M}$ we define the following variables. To that end, let $\mathcal{P} \in \mathcal{K}_C \times \mathcal{K}_C$ be the set of all consecutive labels and let $s_1, \ldots, s_n$ be the stops of C in that particular order.

$$\begin{aligned} y_{\ell,\ell'} &\in \{0,1\}, && \forall (\ell,\ell') \in \mathcal{P}. \\ z_i &\in \{0,1\}, && 1 \leq i < n \\ h_{i,j} &\in \{0,1\}, && 1 \leq i < n \text{ and } i < j < n \end{aligned}$$

If $y_{\ell,\ell'} = 1$, we interpret it such that both ℓ and ℓ' are selected for the labeling. If $z_i = 1$, we interpret it such that the selected labels of the stops s_i and s_{i+1} form a switchover. If $h_{i,j} = 1$, we interpret it such that the selected labels at s_i, s_{i+1}, s_j and s_{j+1} form two consecutive switchovers. Further, for each metro line C we introduce the following constraints. In these constraints $L(\mathcal{K})$ denotes the set of labels that lie to the left of C, and $R(\mathcal{K})$ denotes the set of labels that lie to the right of C.

$$x_\ell + x_{\ell'} - 1 \leq y_{\ell,\ell'} \qquad \forall (\ell,\ell') \in \mathcal{P} \tag{9.4}$$

$$\sum_{\ell \in L(\mathcal{K}_{s_i})} x_\ell + \sum_{\ell \in R(\mathcal{K}_{s_{i+1}})} x_\ell - 1 \leq z_i \qquad 1 \leq i < n \tag{9.5}$$

$$\sum_{\ell \in R(\mathcal{K}_{s_i})} x_\ell + \sum_{\ell \in L(\mathcal{K}_{s_{i+1}})} x_\ell - 1 \leq z_i \qquad 1 \leq i < n \tag{9.6}$$

$$z_i + z_j - 1 - \sum_{i<k<j} z_k \leq h_{i,j} \qquad 1 \leq i < j < n \tag{9.7}$$

We further define for each metro line C the following linear term.

$$w(C) := \sum_{\ell \in \mathcal{K}_C} x_\ell \cdot w_1(\ell) + \sum_{(\ell,\ell') \in \mathcal{P}} y_{\ell,\ell'} \cdot w_2(\ell,\ell') + \sum_{\substack{1 \leq i < n \\ i<j<n}} h_{i,j} \cdot w_3(s_i, s_j) \tag{9.8}$$

Subject to the presented Constraints (9.1)–(9.7) we then minimize

$$\sum_{C \in \mathcal{M}} w(C). \tag{9.9}$$

Consider a variable assignment that minimizes (9.9) and satisfies Constraints (9.1)–(9.7). We show that

$$\mathcal{L} = \{\ell \in \mathcal{K} \mid x_\ell = 1\}$$

is an optimal labeling of the given instance with respect to the given cost function. First of all, $\mathcal{L}$ is a valid labeling. Constraint (9.1) ensures that no label in $\mathcal{L}$ intersects any metro line. By Constraint (9.2) the labels in $\mathcal{L}$ are pairwise disjoint. Finally, by Constraint (9.3) for each stop there is exactly one label contained in $\mathcal{L}$. In particular, for a metro line $C \in \mathcal{M}$, the set $\mathcal{L}_C = \mathcal{L} \cap \mathcal{K}_C$ is a valid labeling of C.

We now show that $w(\mathcal{L}_C) = w(C)$ for any metro line $C \in \mathcal{M}$. Since we minimize (9.9), this implies the optimality of $\mathcal{L}$. Obviously, for a label $\ell \in \mathcal{K}_C$ the cost $w_1(\ell)$ is taken into account in $w(C)$ if and only if ℓ belongs to $\mathcal{L}$.

By Constraint (9.4) for two consecutive labels ℓ and ℓ' of C we have $y_{\ell,\ell'} = 1$ if both are contained in $\mathcal{L}$. Further, by the minimality of (9.9), if at least one of both labels does not belong to $\mathcal{L}$, it holds $y_{\ell,\ell'} = 0$. Hence, $w_2(\ell, \ell')$ is taken into account in $w(C)$ if and only if both ℓ and ℓ' belong to $\mathcal{L}$.

By Constraint (9.5) and Constraint (9.6) it holds $z_{i,j} = 1$ if the labels $\ell_i \in \mathcal{L}$ and $\ell_{i+1} \in \mathcal{L}$ of the consecutive stops s_i and s_{i+1} form a switchover in $\mathcal{L}$. Hence, by Constraint (9.7) it further holds $h_{i,j} = 1$ if the labels of s_i and s_{i+1} as well as s_j and s_{j+1} form switchovers σ_i and σ_j in $\mathcal{L}$, and, furthermore, there is no other switchover in between σ_i and σ_j, i.e., both switchovers are consecutive. On the other hand, by the minimality of $w(C)$ in all other cases it holds $h_{i,j} = 0$. Hence, $w_3(s_i, s_j)$ is taken into account in $w(C)$ if and only if σ_i and σ_j are consecutive switchovers in $\mathcal{L}$. Altogether we obtain the following theorem.

Theorem 9.1. *Given an optimal variable assignment for the presented ILP formulation, the set $\mathcal{L} = \{\ell \in \mathcal{K} \mid x_\ell = 1\}$ is an optimal labeling of $(\mathcal{M}, \mathcal{S}, \mathcal{K})$ with respect to w.*

The approach IlpAlg simply replaces the second phase of DpAlg by that integer linear programming formulation. Hence, it solves the second phase optimally. The approach ScaleAlg samples a predefined scaling range $[x_{min}, x_{max}]$, which is also used by DpAlg. For each scale x it scales the candidates correspondingly. Using the ILP formulation it then checks whether the candidates admit a labeling. Hence, we approximately obtain the greatest scaling factor that admits a labeling.

9.3.2 Greedy-Algorithm

The algorithm GreedyAlg replaces the dynamic programming approach in our workflow as follows. Starting with the solution $\mathcal{L}$ enforced by the 1st step, the greedy algorithm iterates once through the stops of C. For each stop s of C it selects the candidate $\ell \in \mathcal{K}_s$ that minimizes $w_1(\ell) + w_2(\ell_p, \ell) + w_2(\ell, \ell_s)$ among all valid candidates in $\mathcal{K}_s$, where $\ell_p \in \mathcal{L}$ is the candidate selected for the previous stop s_p and $\ell_s \in \mathcal{L}$ is the candidate for the successive stop. It replaces the candidate of s in $\mathcal{L}$ with ℓ.

Instance	*Style*	$\frac{s}{s_{max}}$	*Reference*
Sydney1	Octilinear	0.69	[NW11, Figure 9a]
Sydney2	Octilinear	0.57	[NW11, Figure 9b]
Sydney3	Octilinear	0.57	[WC11, Figure 10.]
Sydney4	Curved	0.67	[Fin+13, Figure 1a]
Vienna1	Octilinear	0.59	[NW11, Figure13a]
Vienna2	Octilinear	0.81	[NW11, Figure 13b]
Vienna3	Curved	0.54	[Fin+13, Figure 8c]

Table 9.1: Overview of considered instances. *Style:* Style of map and applied labels. $\frac{s}{s_{max}}$: Ratio of applied scale factors. The scale s_{max} is a lower bound for the largest possible scaling factor (obtained by ScaleAlg), and s is the scale computed by the first phase of our workflow.

9.4 Evaluation

To evaluate our approach presented in Section 9.2, we did a case study on the metro systems of Sydney (173 stops) and Vienna (84 stops), which have been used as benchmarks before [Fin+13, NW11, WC11]. For Sydney we took the curved layout from [Fin+13, Figure 1a] and the octilinear layouts from [NW11, Figure 9a,b][WC11, Figure 10.], while for Vienna we took the curved layout from [Fin+13, Figure 8c] and the ocitlinear layouts from [NW11, Figure 13a,b]. See also Table 9.1 for an overview of the instances. Since the metro lines of Sydney are not only paths, we disassembled those metro lines into single paths. We did this by hand and tried to extract as long paths as possible. Hence, the instances of Sydney decompose into 12 lines and the instances of Vienna into 5 lines. We took the positions of the stops as presented in the corresponding papers. In the curved layout of Sydney we removed the stops *Tempe* and *Martin Place* (in Figure 9.2(a) marked as red dots), because both stops are tightly enclosed by metro lines such that only the placement of very small labels is possible. This is not so much a problem of our approach, but of the given layout. In a semi-automatic approach the designer would then need to change the layout. For the curvilinear layouts we used labels of CurvedStyle and for the octilinear layouts we used labels of OctilinStyle. For the layouts of *Sydney2*, *Sydney3* and *Vienna2* the authors present labelings; see Figure 9.3, Figure 9.4 and Figure 9.5. For any other layout they do not present labelings.

The experiments were performed on a single core of an Intel(R) Core(TM) i7-3520M CPU processor. The machine is clocked at 2.9 GHz, and has 4096 MB RAM. Our implementation is written in Java. For each instance and each algorithm we conducted 100 runs and took the average running times. Each time before we started the 100 runs, we performed 50 runs without measuring the running time in order to *warm up* the virtual machine (Java(TM) SE Runtime Environment, build 1.8.0_60-b27, Oracle).

Table 9.2: Running times in seconds of the workflow broken down into its two phases and their single steps (if applicable). *Algo.*: The applied algorithm. Times less than 0.01 seconds are marked with ⋆.

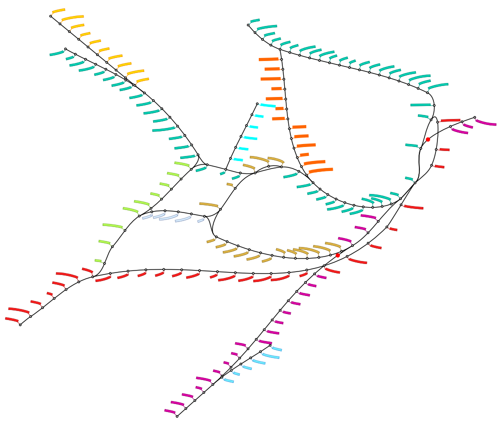
		Algo.	*Phase 1: Creation*			*Phase 2: Selection*			
Instance	Layout		*1st*	*2nd*	∑	*1st*	*2nd*	∑	**Total**
Sydney1	Octi.	DpAlg	0.03	0.18	0.21	0.09	0.1	0.18	**0.39**
		GreedyAlg	0.02	0.18	0.2	⋆	⋆	⋆	**0.2**
		ScaleAlg	0.02	–	0.02	–	–	69.09	**69.11**
		IlpAlg	0.02	0.17	0.19	–	–	25.43	**25.62**
Sydney2	Octi.	DpAlg	0.03	0.28	0.31	0.07	0.09	0.15	**0.46**
		GreedyAlg	0.02	0.28	0.3	⋆	⋆	⋆	**0.3**
		ScaleAlg	0.02	–	0.02	–	–	17.29	**17.31**
		IlpAlg	0.02	0.28	0.29	–	–	15.17	**15.46**
Sydney3	Octi.	DpAlg	0.03	0.17	0.2	0.07	0.09	0.16	**0.36**
		GreedyAlg	0.02	0.16	0.17	⋆	⋆	⋆	**0.17**
		ScaleAlg	0.02	–	0.02	–	–	33.61	**33.63**
		IlpAlg	0.02	0.15	0.17	–	–	21.56	**21.73**
Sydney4	Curved	DpAlg	0.03	0.33	0.36	0.1	0.1	0.2	**0.56**
		GreedyAlg	0.03	0.33	0.35	⋆	⋆	⋆	**0.35**
		ScaleAlg	0.03	–	0.05	–	–	243.58	**243.63**
		IlpAlg	0.02	0.34	0.36	–	–	98.81	**99.17**
Vienna1	Octi.	DpAlg	0.01	0.06	0.07	0.01	⋆	0.02	**0.09**
		GreedyAlg	0.01	0.06	0.07	⋆	⋆	⋆	**0.07**
		ScaleAlg	0.01	–	0.01	–	–	3.31	**3.32**
		IlpAlg	0.01	0.06	0.08	–	–	0.55	**0.63**
Vienna2	Octi.	DpAlg	0.01	0.09	0.1	⋆	⋆	⋆	**0.11**
		GreedyAlg	0.01	0.1	0.11	⋆	⋆	⋆	**0.11**
		ScaleAlg	0.01	–	0.01	–	–	4.65	**4.66**
		IlpAlg	0.01	0.08	0.09	–	–	0.36	**0.45**
Vienna3	Curved	DpAlg	0.02	0.15	0.17	0.02	0.04	0.06	**0.23**
		GreedyAlg	0.02	0.14	0.16	⋆	⋆	⋆	**0.16**
		ScaleAlg	0.02	–	0.02	–	–	7.27	**7.29**
		IlpAlg	0.01	0.13	0.15	–	–	0.49	**0.64**

Table 9.3: Experimental results for Sydney. *Algo.*: The applied algorithm. *Candidates*: Values concerning candidates; No. of candidates after the first and third step, *A12*= No. of labels removed to establish Assumption 8.1 and Assumption 8.2, SO= No. of switchovers. *Cost*: Ratio of costs; $\mathcal{L}_A$ = labeling obtained by procedure $A \in \{$DpAlg, GreedyAlg, IlpAlg, ScaleAlg$\}$, $\mathcal{L}$ = labeling obtained by procedure IlpAlg. *Sequence*: Values concerning sequences of labels lying on the same side of their metro line; *min*=length of shortest sequence, *max*=length of longest sequence, *avg.*=average length of sequences.

			Candidates				Cost				Sequence		
Instance	Layout	Algo.	1st	3rd	A12	SO	$\frac{w_1(\mathcal{L}_A)}{w_1(\mathcal{L})}$	$\frac{w_2(\mathcal{L}_A)}{w_2(\mathcal{L})}$	$\frac{w_3(\mathcal{L}_A)}{w_3(\mathcal{L})}$	$\frac{w(\mathcal{L}_A)}{w(\mathcal{L})}$	min	max	avg.
Sydney1	Octi.	DpAlg	1164	1004	9	4	1.73	1.12	0.63	1.16	3	41	9.53
		GreedyAlg	1164	1005	9	15	1.73	1.87	7.62	2.64	1	24	5.81
		ScaleAlg	1164	–	–	13	2.46	1.67	2.91	1.97	3	20	6.97
		IlpAlg	1164	–	–	7	1	1	1	1	3	20	8.1
Sydney2	Octi.	DpAlg	1104	963	3	6	1.36	1.08	3.25	1.33	1	26	9.34
		GreedyAlg	1104	964	3	19	2.46	1.83	19.52	3.54	1	35	5.94
		ScaleAlg	1104	–	–	15	3.91	1.92	8.88	2.91	1	19	6.76
		IlpAlg	1104	–	–	5	1	1	1	1	3	25	9.48
Sydney3	Octi.	DpAlg	1132	1013	0	6	1	1	1	1	3	25	8.76
		GreedyAlg	1132	1013	0	14	2.15	1.79	5.99	2.61	1	23	6.9
		ScaleAlg	1132	–	–	12	2.72	2.68	3.06	2.76	2	21	7.02
		IlpAlg	1132	–	–	6	1	1	1	1	3	25	8.76
Sydney4	Curved	DpAlg	1275	1020	2	8	1	1.13	1.15	1.01	3	27	7.38
		GreedyAlg	1275	1021	2	28	1	1.08	12.81	1.16	1	24	4.82
		ScaleAlg	1275	–	–	13	1.05	1.69	2.28	1.1	2	20	6.21
		IlpAlg	1275	–	–	7	1	1	1	1	3	27	7.92

Table 9.4: Experimental results for Vienna. *Algo.*: The applied algorithm. *Candidates*: Values concerning candidates; No. of candidates after the first and third step, *A12*= No. of labels removed to establish Assumption 8.1 and Assumption 8.2, SO= No. of switchovers. *Cost*: Ratio of costs; $\mathcal{L}_A$ = labeling obtained by procedure $A \in \{$DpAlg, GreedyAlg, IlpAlg, ScaleAlg$\}$, $\mathcal{L}$ = labeling obtained by procedure IlpAlg. *Sequence*: Values concerning sequences of labels lying on the same side of their metro line; *min*=length of shortest sequence, *max*=length of longest sequence, *avg.*=average length of sequences.

			Candidates				Cost				Sequence		
Instance	Layout	Algo.	1st	3rd	A12	SO	$\frac{w_1(\mathcal{L}_A)}{w_1(\mathcal{L})}$	$\frac{w_2(\mathcal{L}_A)}{w_2(\mathcal{L})}$	$\frac{w_3(\mathcal{L}_A)}{w_3(\mathcal{L})}$	$\frac{w(\mathcal{L}_A)}{w(\mathcal{L})}$	min	max	avg.
Vienna1	Octi.	DpAlg	466	403	0	2	0.74	1.29	0.99	1.07	8	21	12.8
		GreedyAlg	466	403	0	7	1.26	2.43	8.59	2.48	1	21	9.41
		ScaleAlg	466	–	–	6	2.04	3.14	6.22	2.99	1	19	10.05
		IlpAlg	466	–	–	2	1	1	1	1	8	21	12.8
Vienna2	Octi.	DpAlg	468	324	1	8	2.63	1.03	4.01	1.52	1	16	8.12
		GreedyAlg	468	324	1	18	4.79	1.72	14.23	3.36	1	12	4.31
		ScaleAlg	468	–	–	6	2.63	1.07	2.55	1.4	1	12	7.87
		IlpAlg	468	–	–	4	1	1	1	1	5	16	10
Vienna3	Curved	DpAlg	606	535	0	7	1	1	1	1	2	22	10.7
		GreedyAlg	606	535	0	11	0.98	1.47	1.92	1.06	1	17	6.6
		ScaleAlg	606	–	–	13	1.18	3.2	2.18	1.34	1	16	7.43
		IlpAlg	606	–	–	7	1	1	1	1	2	22	10.7


(a) *Sydney4*: DpAlg, CurvedStyle Layout from [Fin+13, Figure 1a.].

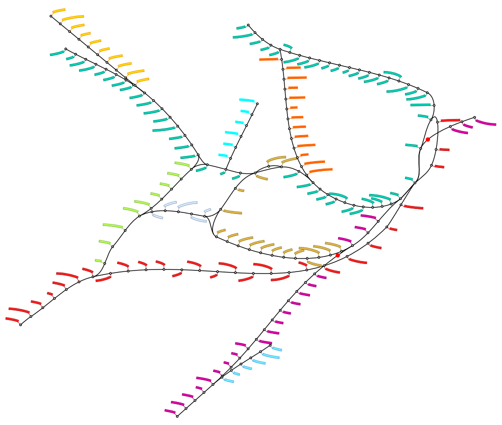

(b) *Sydney4*: GreedyAlg, CurvedStyle Layout from [Fin+13, Figure 1a].

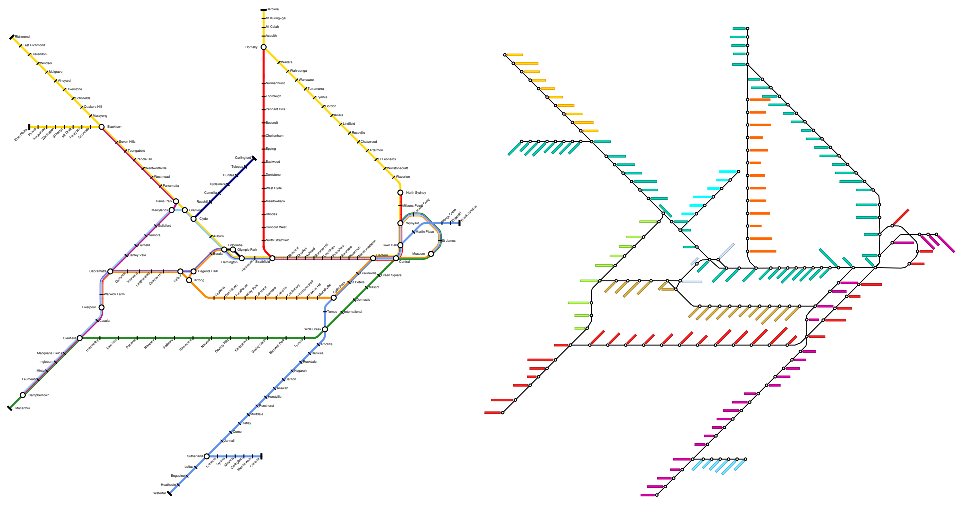
Figure 9.2: Labelings for the metro map of Sydney using DpAlg and GreedyAlg.

We did this in order to measure the actual running times of our algorithms and not to measure the time that the virtual machine of Java spends for loading classes and optimizing byte code.

Table 9.2, Table 9.3 and Table 9.4 present our quantitative results for the considered instances. For *Sydney4* labelings are found in Figure 9.2 and for *Sydney2*, *Sydney3* and *Vienna2* labelings created by DpAlg are found in Figure 9.3, Figure 9.4, Figure 9.5, respectively. The labelings of all instances are found in Section 9.6.

We first note that with respect to the total number of created candidates only few labels are removed for enforcing Assumption 8.1 and Assumption 8.2; see Table 9.3 and Table 9.4, *A12*. This indicates that requiring those assumptions is not a real restriction on a realistic set of candidates, even though they seem to be artificial.

Running Time. Even for large networks as Sydney, our algorithm DpAlg needs less than 0.6 seconds; see Table 9.2. This shows that our approach is applicable for scenarios in which the map designer wants to adapt the layout and its labeling interactively. In particular, in those scenarios not every of the four steps must be repeated each time, which improves computing time. For example, after once applying the scaling step (1st phase, 2nd step – the most time consuming step), the instance does not need to be rescaled again, but the relation between label size and map size is determined. Further, DpAlg is only moderately slower than GreedyAlg; 0.21 seconds in maximum, see Table 9.2, *Sydney4*. On the other hand, the approaches IlpAlg and ScaleAlg are not alternatives, because their running times are much worse; over 1 minute in maximum; see Table 9.2, *Sydney4*.



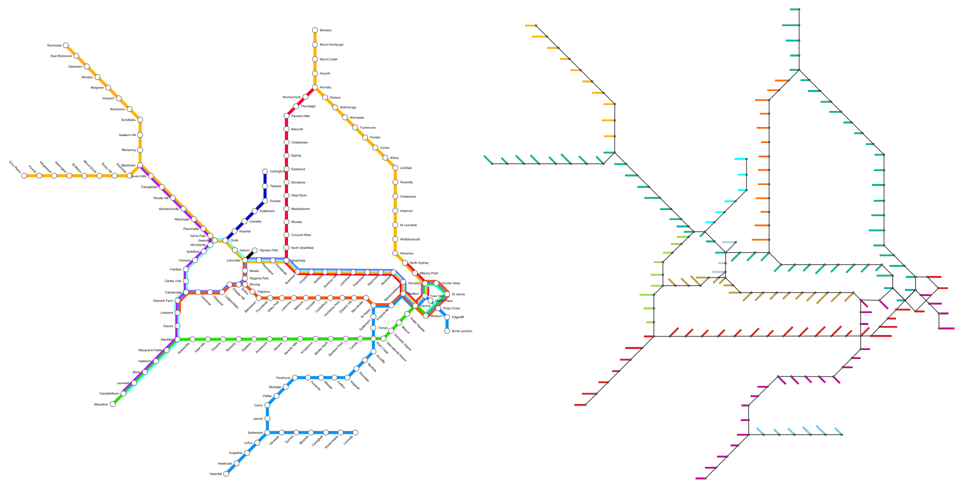
(a) Original labeling by [NW11]

(b) *Sydney2*: DpAlg, OctilinStyle Layout from [NW11, Figure 9b.].

Figure 9.3: Labelings for the metro map of Sydney.

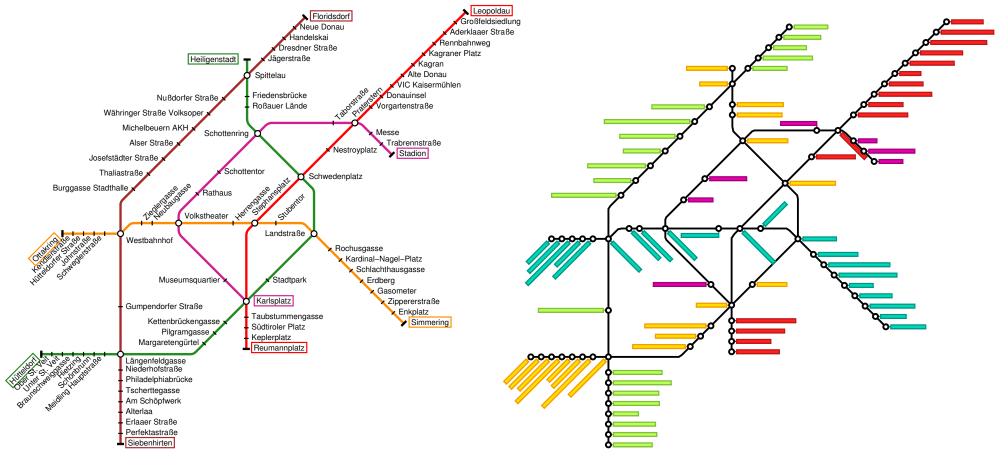
Quality. We observe that in all labelings created by DpAlg there are only few switchovers, namely 4–8; see Table 9.3 and Table 9.4, column SO. Hence, there are long *sequences* of consecutive labels that lie on the same side of their metro line; see corresponding figures and Table 9.3 and Table 9.4, column Sequence. Together with the ILP based approach IlpAlg, it yields the solution with the longest sequences in average. In particular the switchovers are placed such that those sequences are regularly sized. The labels of a single sequence are mostly directed into the same x-direction and in particular they are similarly shaped so that those sequences of labels form regular patterns as desired. The alignment of the labels is chosen so that they blend in with the alignment of their adjacent labels. In comparison with the solution of IlpAlg, the costs of DpAlg never exceed a factor of 1.52; see Table 9.3 and Table 9.4, column $\frac{w(\mathcal{L}_{\text{DpAlg}})}{w(\mathcal{L})}$. For the instances *Sydney4* and *Vienna3* it even obtains a solution with the same costs. For the other instances, DpAlg basically spends its additional costs on the choice of the single labels ($\frac{w_1(\mathcal{L}_{\text{DpAlg}})}{w_1(\mathcal{L})}$) and the distance of switchovers ($\frac{w_3(\mathcal{L}_{\text{DpAlg}})}{w_3(\mathcal{L})}$).

In contrast, GreedyAlg yields significantly more switchovers; in maximum 20 switchovers more than DpAlg, see *Sydney4* . Consequently, there are many distracting switches of labels from one side to the other of the metro line; e.g. see Figure 9.2. Although the sequences of consecutive labels lying on the same side may be longer in maximum compared to DpAlg, they are much shorter in average; see Table 9.3 and Table 9.4, column Sequence. Further, several adjacent labels point in opposite x-directions, which results in distracting effects; see corresponding figures. Altogether, the labelings that are obtained by GreedyAlg do not look regular, but cluttered. DpAlg



(a) **(b)**

Figure 9.4: Labelings for Sydney. (a) Original Labeling presented by Wang and Chi [WC11] (b) *Sydney3*: DpAlg.



(a) **(b)**

Figure 9.5: Labelings for Vienna. (a) Original Labeling presented by Nöllenburg and Wolff [NW11] (b) *Vienna2*: DpAlg.

solves those problems since it considers the metro line *globally* yielding an optimal labeling for a single line. This observation is also reflected in Table 9.3 and Table 9.4, column $\frac{w(\mathcal{L}_{\text{DpAlg}})}{w(\mathcal{L})}$, which shows that the costs computed by GreedyAlg are significantly larger than the costs computed by DpAlg. In particular costs for positioning the switchovers are much worse; *Sydney2*: $w_3(\mathcal{L}_{\text{GreedyAlg}})/w_3(\mathcal{L}) = 19.52$, $w_3(\mathcal{L}_{\text{DpAlg}})/w_3(\mathcal{L}) =$

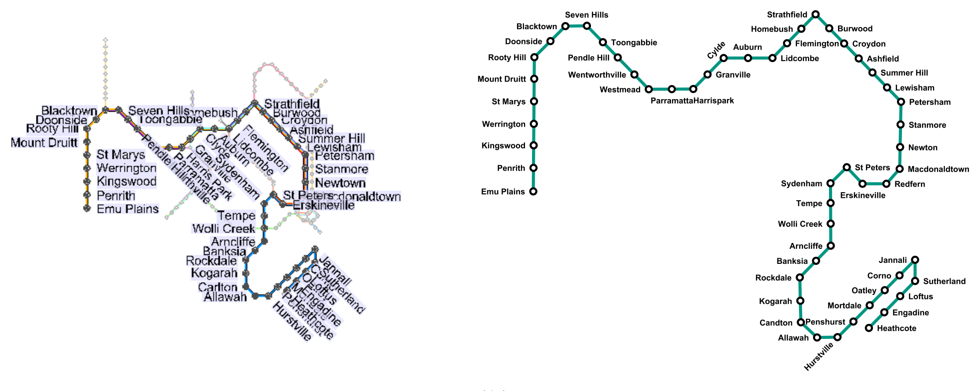


Figure 9.6: Comparison of two labelings for the same line. (a) Labeling is created by the tool presented by Wang and Chi[WC11]. (b) Labeling is created by DpAlg.

3.25 and *Vienna2*: $w_3(\mathcal{L}_{\text{GreedyAlg}})/w_3(\mathcal{L}) = 14.23$, $w_3(\mathcal{L}_{\text{DpAlg}})/w_3(\mathcal{L}) = 4.01$. Hence, the better quality of DpAlg prevails the slightly better running time of GreedyAlg.

Concerning the computed scale factor in the first phase of DpAlg, the labels are smaller than those produced by ScaleAlg by a factor of 0.54–0.81; see Table 9.1. While this seems to be a drawback on the first sight, the smaller size provides necessary space that is used to obtain a labeling of higher quality with respect to the number and the placement of switchovers. Hence, the solutions of ScaleAlg have more switchovers (except for *Vienna2*) and shorter sequences of labels lying on the same side in average than DpAlg; see Table 9.3 and Table 9.4, column SO and Sequence.

We observe that both Nöllenburg and Wolff's and our labelings of Sydney look quite similar, whereas our labeling has less switchovers; see Figure 9.3. The same applies for the labelings of the layout of Vienna; see Figure 9.5. Recall that their approach needed more than 10 hours to compute a labeled metro map of Sydney. Since they need only up to 23 minutes to compute the layout without labeling, it lends itself to first apply their approach to gain a layout and then to apply our approach to construct a corresponding labeling.

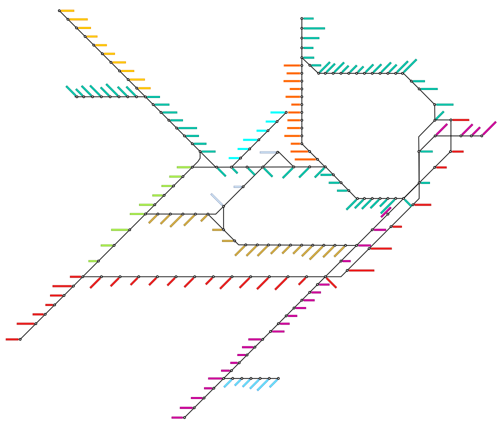
Wang and Chi present in their paper [WC11] an approach that is divided into two phases. In the first phase they compute the layout of the metro map and then in the second phase they create a labeling for that layout. For both steps they formulate energy functions expressing their desired objectives, which then are locally optimized. Figure 9.4(a) shows the metro map of Sydney created by their approach. In comparison, Figure 9.4(b) shows the same layout with a labeling created by our approach. Both labelings look quite similar. While our approach needed 0.26s (see Table 9.2, *Sydney3*), their approach needed less than 0.1s on their machine. However, their approach does not guarantee that the labels are occlusion-free, but labels may overlap with metro lines and other labels. This may result in illegible drawings of metro maps. For example Figure 9.6 shows two labelings of a metro line of Sydney that has been laid out by the tool of Wang and Chi. Figure 9.6(a) shows a labeling that has been created by their tool, while Figure 9.6(b) shows a labeling that has been created by our approach. The labeling of Wang and Chi has several serious defects that makes the map hardly readable. The marked regions A, B and C show labels that overlap each other. Hence, some of the labels are obscured partly, while some of the labels are completely covered by other labels. For example in region B the label *St. Peters* and the label *Erskinville* overlap the label *Macdonaldtown* such that it is hardly viewable. Further, region C contains two diagonal rows of stops aligned parallel. While the upper row is visible, the lower row is almost completely covered by labels. Further, the labels of the upper row obscure the labels of the lower row. In contrast our approach yields an occlusion-free labeling, such that each label and each stop is easily legible. We therefore think that our approach is a reasonable alternative for the labeling step of Wang and Chi's approach. In particular, we think that the better quality of our approach prevails the better running time of Wang and Chi's approach.

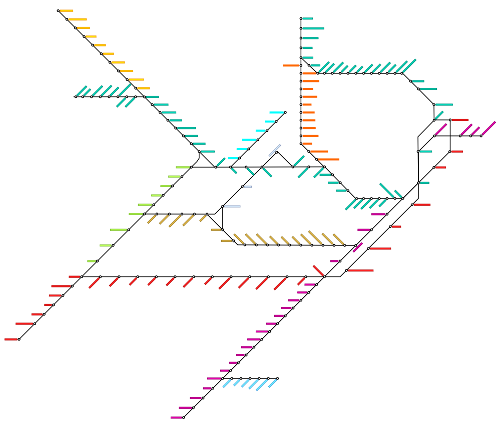
In conclusion our workflow is a reasonable alternative and improvement for the approaches presented both by Nöllenburg and Wolff, and by Wang and Chi. In the former case, our approach is significantly faster, while in contrast to the latter case we can guarantee occlusion-free labelings.

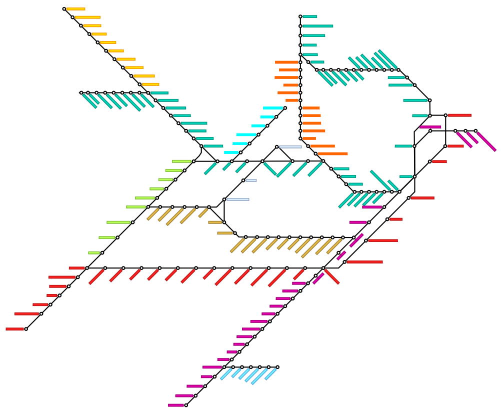
9.5 Conclusions

In this chapter, we presented a sophisticated workflow for labeling a complete metro map. At the core of the workflow we used an efficient algorithm that labels a single metro line optimally with respect to a given objective function. For comparison we developed alternative approaches comprising both integer linear programming formulations as well as simple greedy algorithms. In a detailed evaluation, we showed that our approach yields near optimal labelings. By comparison with a simple greedy algorithm, we further justified the use of the dynamic programming approach; the greedy algorithm produces labelings of significantly lower quality. Altogether, we presented a flexible workflow for labeling metro maps that can be used in automatic as well as semi-automatic settings.

9.6 Labelings


(a) DpAlg


(b) GreedyAlg


(c) ScaleAlg

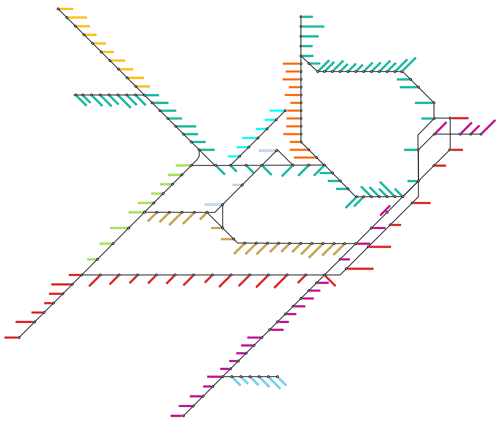
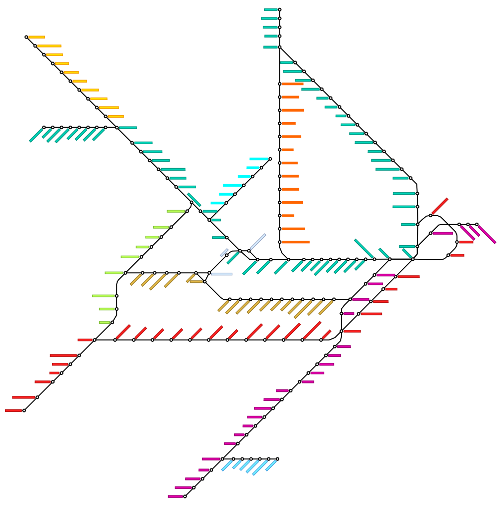
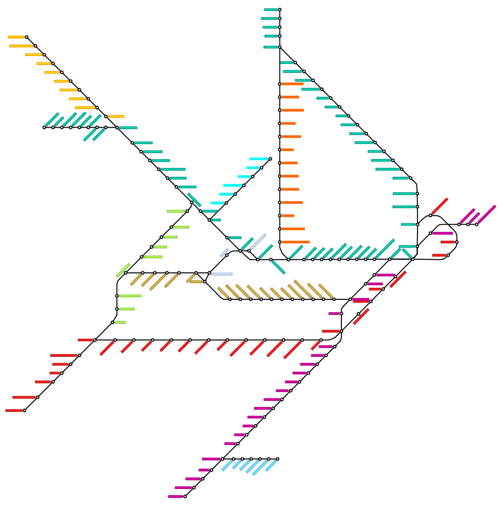
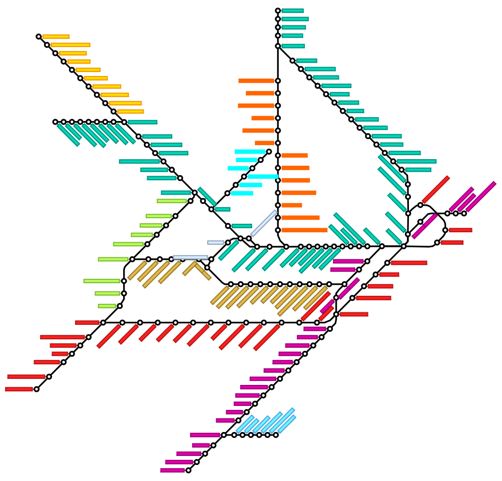

(d) IlpAlg

Figure 9.7: Labelings for instance *Sydney1*.


(a) DpAlg


(b) GreedyAlg


(c) ScaleAlg

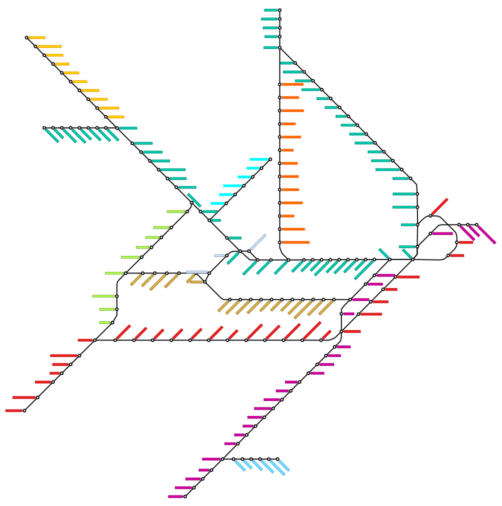
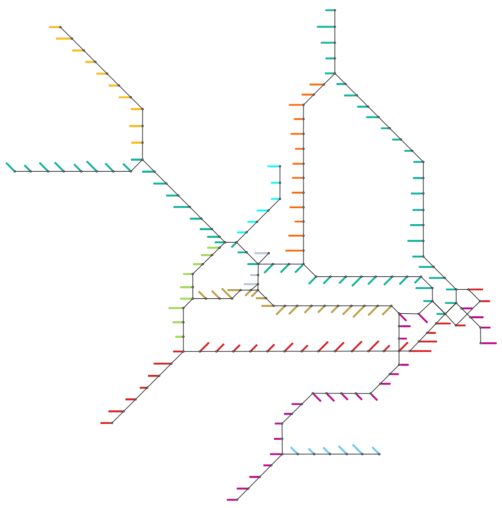
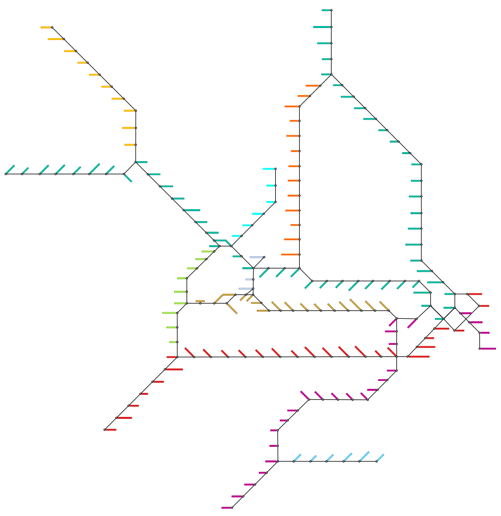
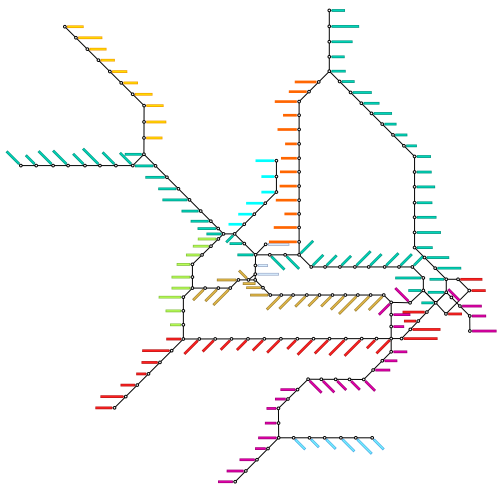

(d) IlpAlg

Figure 9.8: Labelings for instance *Sydney2*.


(a) DpAlg


(b) GreedyAlg


(c) ScaleAlg

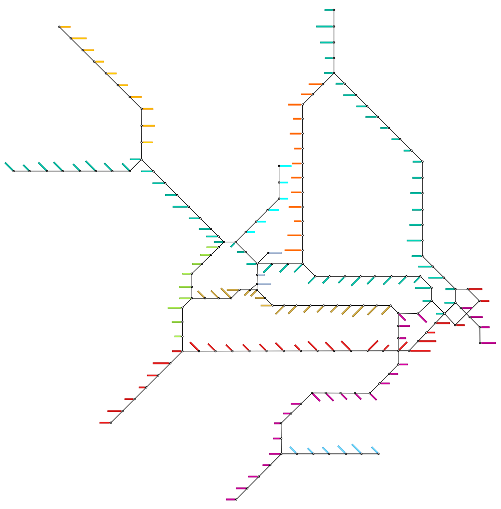
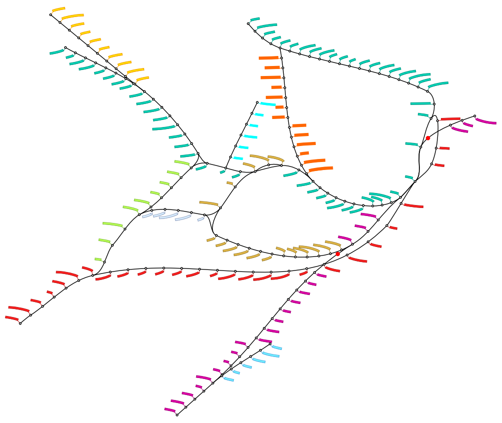
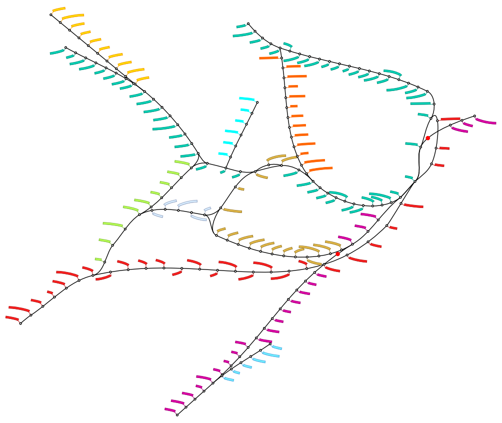

(d) IlpAlg

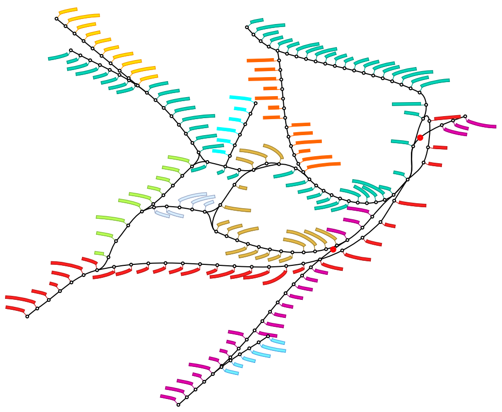
Figure 9.9: Labelings for instance *Sydney3*.



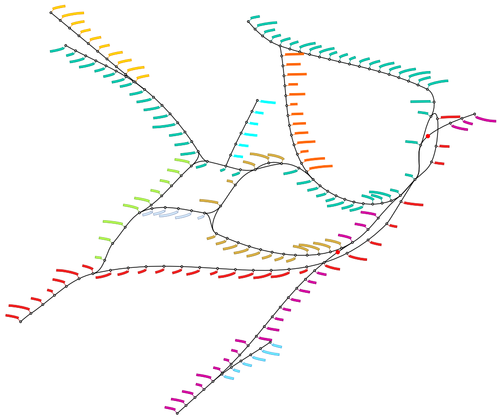
(a) DPALG



(b) GREEDYALG

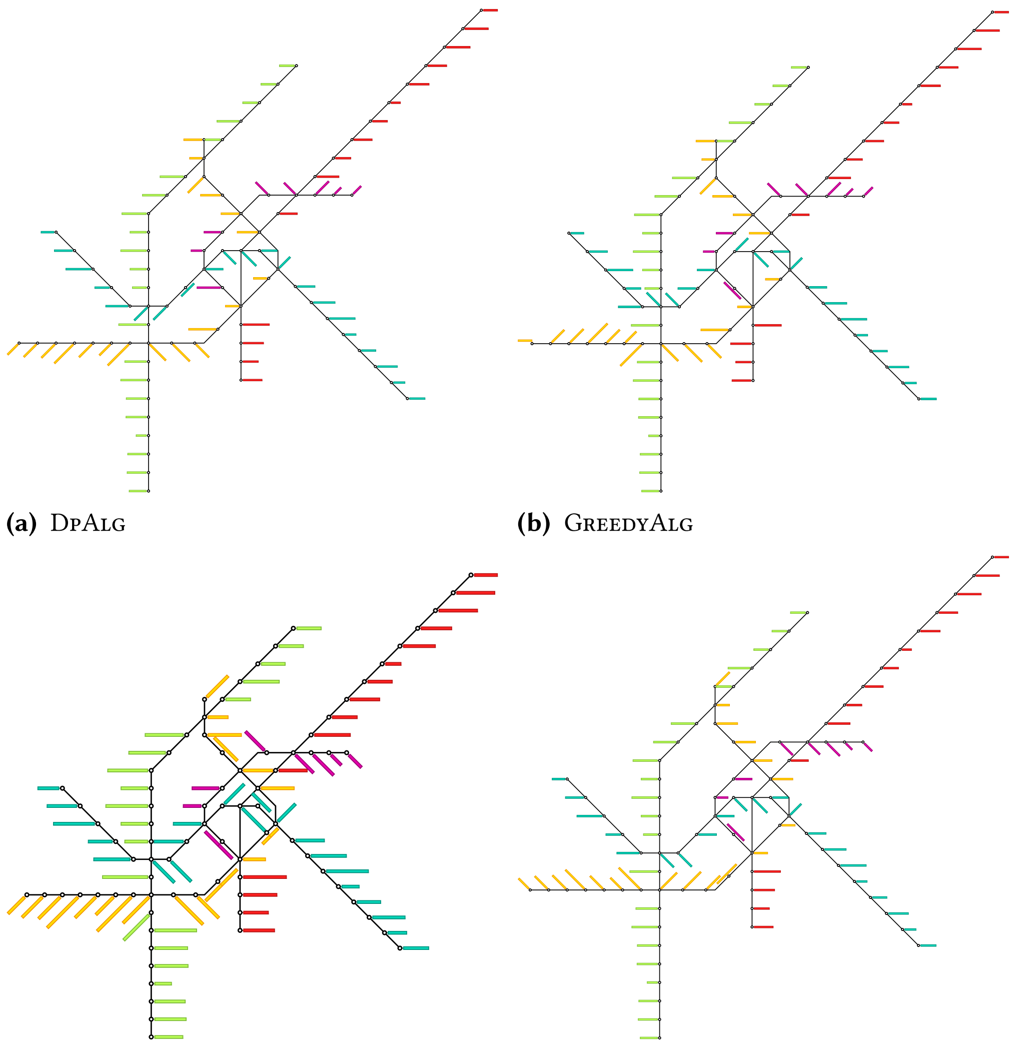


(c) SCALEALG



(d) ILPALG

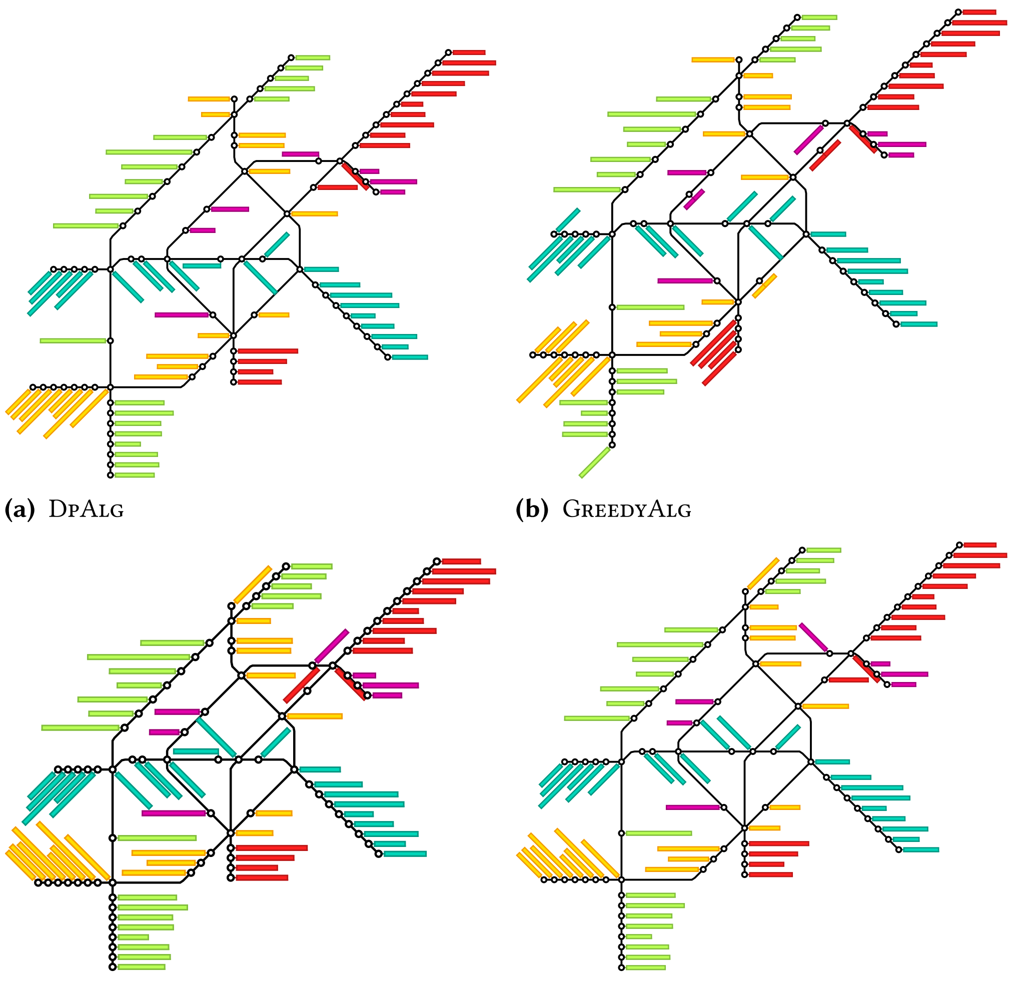
Figure 9.10: Labelings for instance *Sydney4*.



(a) DpAlg **(b)** GreedyAlg

(c) ScaleAlg **(d)** IlpAlg

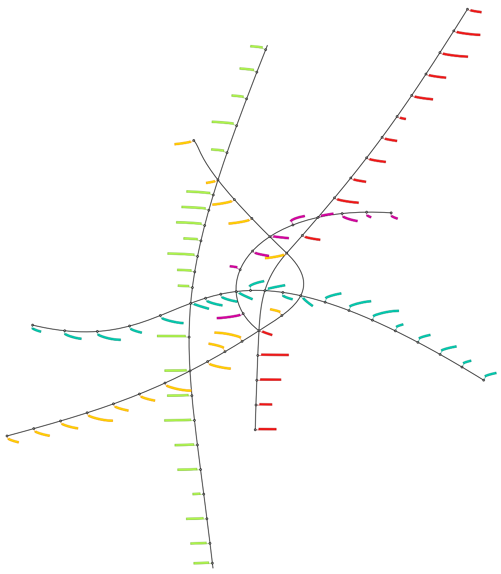
Figure 9.11: Labelings for instance *Vienna1*.



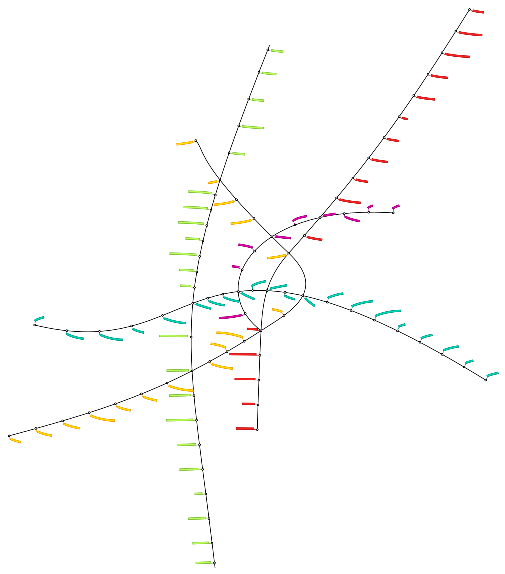
(a) DpAlg (b) GreedyAlg

(c) ScaleAlg (d) IlpAlg

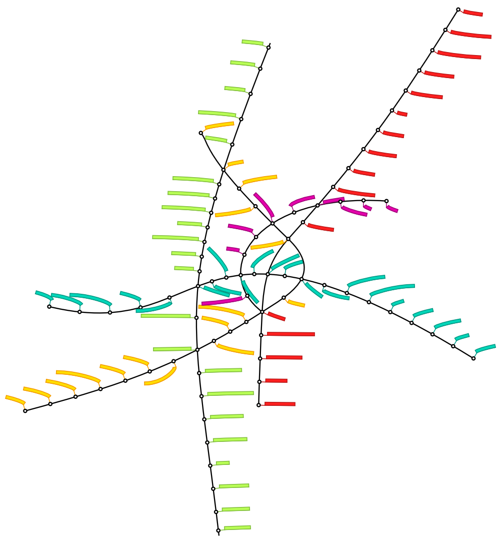
Figure 9.12: Labelings for instance *Vienna2*.



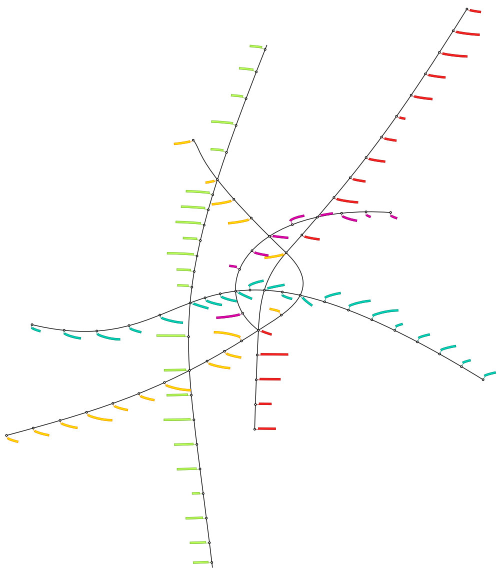
(a) DpAlg



(b) GreedyAlg



(c) ScaleAlg



(d) IlpAlg

Figure 9.13: Labelings for instance *Vienna3*.

Part II

External Label Placement

10 Introduction to External Label Placement

In some applications internal label placement is not sufficient because either features are distributed in dense clusters and there are too many labels to be placed or any extensive occlusion of the figure's details should be avoided. While in the first case one may exclude less important labels, in the second case even a small number of labels may spoil the readability of the figure. In either case, graphic designers often choose to place *external* labels outside of the figure and connect features with their labels by thin curves, so-called *leaders*. This kind of labeling is commonly found in highly detailed scientific figures as they are used, for instance, in atlases of human anatomy, see Figure 10.1(a) for a historical example, as well as for annotations in infographics in media, see Figure 10.1(b) for a map in television.

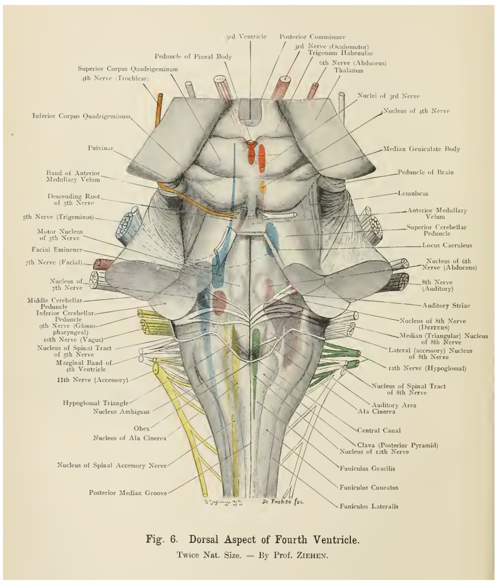
The automated placement of external labels has been studied extensively both from a practical and formal, i.e., algorithmic, perspective, often using a slightly different terminology and a variety of models, constraints, and objectives. In the remainder of this chapter, we introduce a unifying terminology and give a structured overview of the related work.

This chapter is based on and partly taken from joint work with Lukas Barth, Andreas Gemsa and Martin Nöllenburg [Bar+15].

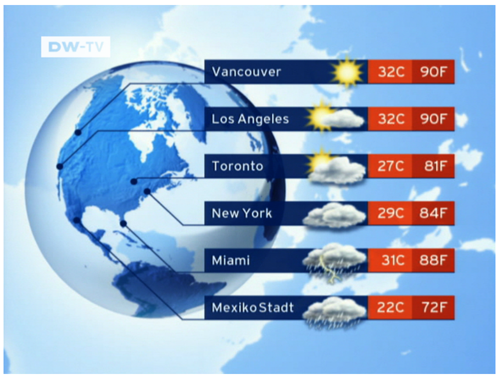
10.1 Models of External Labeling

In *external point labeling*, the input is typically a figure contained in some bounding polygon, together with a finite set of points within the polygon; we call these points *sites*. Each site s is assigned to a text that describes the site. Using a standard simplification in map labeling, not the text itself is considered, but its shape is approximated by its axis-aligned bounding box ℓ. We call ℓ the *label* of the site s. The general external labeling problem is then to find a non-overlapping placement of all labels of the given sites outside of the bounding polygon and a set of non-crossing leaders connecting each site to its label while satisfying certain additional constraints and optimizing some quality function.

Following Tufte's minimum-ink principle [Tuf01], the most common objective in external labeling is to minimize the total leader length, which means minimizing the total overlay of leaders with the given figure. While the theoretical results typically ask explicitly for the labeling with minimum total leader length, many practical algorithms



(a) Human ventricle.



(b) Weather map.

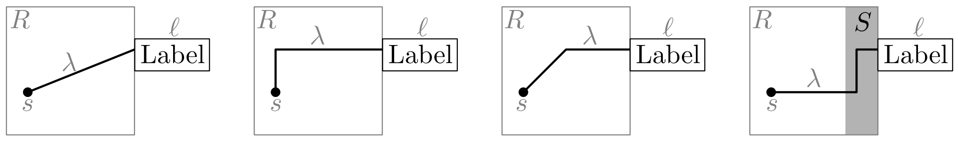
Figure 10.1: (a) Illustration of the human ventricle using external labeling with type-s leaders. Source: *Atlas of applied (topographical) human anatomy for students and practitioners* Authors: *Karl Heinrich von Bardeleben (1849-1919), Ernst Haeckel (1834-1919)* Publication: *London, Rebman Limited; New York, Rebman Company, 1906.* (b) Weather forecast map using boundary labeling with type-do leaders. Image courtesy of Deutsche Welle TV.

are based on heuristic approaches that only implicitly aim for short leaders, but do not necessarily achieve the global minimum length.

A particularly well-studied special case of external labeling is the *boundary labeling problem*, which assumes that the bounding polygon is actually a rectangle R and all labels must be placed touching the boundary of R, or even touching a particular side of R. More precisely, the boundary labeling problem asks for the placement of the labels such that

(1) each label ℓ lies outside R and touches the boundary of R,
(2) no two labels overlap,
(3) for each site s and its label ℓ there is a leader λ in R, i.e., a simple curve in R, that starts at s and ends on the boundary of ℓ, and
(4) no two leaders intersect.

The end point of a leader λ that touches a label ℓ is called the *port* of ℓ. Typically, four main parameters in which the models differ are distinguished. The *label position* specifies on which sides of R the labels are placed. The *label size* may be uniform or individually defined for each label. The *port type* specifies whether *fixed ports* or *sliding ports* are used, i.e., whether the position of a port on its label is pre-defined or



(a) *s*-leader **(b)** *po*-leader **(c)** *do*-leader **(d)** *opo*-leader

Figure 10.2: Illustration of leader types. Type-*opo* leaders use a track routing area S.

flexible. Finally, the *leader type* restricts the shape of the leaders. In the following, we list the four leader types that are most commonly found in the literature.

Let λ be a leader connecting a site s with its label ℓ, and let r be the side of R that is touched by ℓ. An *s-leader* consists of a single straight (s) line segment; see Figure 10.2(a). A *po-leader* consists of two line segments: the first, starting at s, is parallel (p) to r and the second segment is orthogonal (o) to r; see Figure 10.2(b). A *do-leader* consists of two line segments: the first, starting at s, is diagonal (d) at some fixed angle $\pm\alpha$ (typically $\alpha = 45°$) relative to r and the second segment is orthogonal (o) to r; see Figure 10.2(c). An *opo-leader* consists of three line segments: the first, starting at s, is orthogonal (o) to r, the second is parallel (p) to r, and the third segment is orthogonal (o) to r; see Figure 10.2(d). In case that *opo*-leaders are considered, each leader has its two bends in a strip S next to r whose width is large enough to accommodate all leaders with a minimum pairwise distance of the p-segments. The strip S is called the *track-routing area* of R. We call a labeling based on $s/po/do/opo$-leaders an $s/po/do/opo$-labeling. We note that for s-, *po*- and *do*-leaders fixing the position of the site and the port uniquely defines the according leader. For *opo*-leaders only the x-position of the leader's vertical segment is not fixed and depends on the routing in S.

In the following chapters, we adapt this general model for our purposes. As the leader type is the most distinctive feature of the different boundary labeling models in the literature, we examine how this parameter influences the readability in Chapter 11. To that end, we present the first formal user-study on the readability of leaders. In Chapter 12, we consider boundary labeling with *po*-leaders for the case that labels are placed along two or more sides of R. In Chapter 13, we generalize the model of boundary labeling for s-leaders assuming that R is not a rectangle, but a simple polygon describing the contour of the figure; we call this problem *contour labeling*.

10.2 Related Work

In the following, we give an overview of research on external label placement in maps and figures. Due to the vast amount of practical and theoretical results, we split the discussion into two parts accordingly.

Practical Results. We first review research on applied external label placement in chronological order; a more detailed overview of external labeling is given by Oeltze-Jafra and Preim [OP14].

In 1997 Preim et al. [PRS97] employed a prototype for viewing 3-dimensional anatomical models in combination with zooming and external label placement using s-leaders. While the overall structure of the system is discussed in detail, the authors give only little information about the actual placement of the labels and the applied quality criteria. Based on a manual analysis of hand-drawn illustrations (e.g., anatomic atlases), Hartmann et al. [HAS04] as well as Ali et al. [AHS05] list criteria for external labeling concerning readability, ambiguity and aesthetics. Using these criteria, they present force-based heuristics for labeling figures. While Hartmann et al. investigate s-leaders only, Ali et al. also discuss orthogonal leaders with bends as well as different layout styles. Hartmann et al. [Har+05] further refine and summarize the extracted criteria by introducing varying functional and aesthetic metrics for external labeling; both for s-leaders and orthogonal leaders with bends. In the same year, Bruckner and Gröller [BG05] introduced the system VolumeShop for the interactive exploration of volume illustrations. Based on the guidelines of Ali et al. [AHS05], they present a simple labeling algorithm for s-leaders that iteratively exchanges label positions until a certain quality bound is achieved, or a maximum number of iterations is exceeded. Furthermore, Götzelmann et al. [Göt+05] present and experimentally evaluate an approach to annotate 3D models with internal and external labels using s-leaders. For the layout of external labels they again make use of the approach presented by Ali et al. [AHS05].

In 2006 Götzelmann et al. [GHS06] introduced an alternative approach, which sequentially evaluates and places external labels by priority. Fuchs et al. [Fuc+06] investigate external labeling algorithms for mobile applications, which require fast algorithms that also exploit small spaces for label placement.

Vollik et al. [Vol+07] evaluate a local optimization approach based on energy functions expressing the labeling problem and its quality criteria. They use simulated annealing to find a locally optimal solution. Stein and Décoret [SD08] also formalize external labeling as a mathematical optimization problem based on a set of constraints, which they solve by a greedy algorithm without any optimality guarantees.

For the application of surgery planning, Mühler and Preim [MP09] present a framework to automatically annotate 2D slices and 3D reconstructions of segmented structures. To that end, they use external label placement with s-leaders. Čmolík and

Bittner [ČB10] again take a more mathematical approach and formalize the problem by means of fuzzy logic combined with greedy optimization. Mogalle et al. [Mog+12] use external labels with s-leaders for annotating CT images. They also apply a greedy algorithm on a mathematical formalization of the labeling problem.

Tatzgern et al. [TKS13] consider external labeling for explosion diagrams in augmented reality. Based on the algorithm by Hartmann et al. [HAS04] they first create an initial layout. Then they resolve overlapping labels and leader intersections by refining label positions, and also balance the distance between labels. In 2014 Tatzgern et al. [Tat+14] again considered external label placement in augmented reality. However, this time they propose an approach to place external labels in the 3D object space instead of the 2D image space. Madsen et al. [Mad+16] empirically evaluate this setting by conducting a user study showing that, among others, label placement in 3D object space has positive impact on the image's readability. Their study used s-leaders only, since the leader type was not a variable of interest.

Apart from those 16 papers, there is related research about excentric labeling (e.g., [Fek99, BRL09]), and other labelings that use the concept of leaders to associate labels, but without explicitly differentiating between the space for labels and the space for sites (e.g., [AF03, MD06, Pet+09b, Pet+09a]).

The main interest of practical external labeling research is the design and actual implementation of fast external label placement algorithms for a variety of applications. User studies are conducted to evaluate particular models and labeling algorithms. Further, much work is invested into elaborated labeling models that take multiple quality criteria into account. A common quality criterion, though, shared by almost all of the above papers is the minimization of leader lengths in the sense that labels should be placed close to their sites. In order to deal with these complex models and various practical constraints, heuristics are applied instead of exact algorithms; mathematically proven quality guarantees are not the focus of the investigations. Interestingly, there is a clear bias towards s-leaders in the literature. Almost all of the above papers (14 out of 16) exclusively consider s-leaders, the two remaining ones additionally consider orthogonal leaders with bends. Naturally, the question arises, whether the alternative leader types *po*, *do* and *opo*, which are regularly used by graphic designers, are useful in practice and how they compete against s-leaders.

Theoretical Results. Next, we review existing theoretical results on boundary labeling. We again present the related work in chronological order.[1] Boundary labeling was introduced as an algorithmic problem by Bekos et al. [Bek+04, Bek+07] at Graph Drawing 2004. The authors present efficient algorithms for models based on *po*-, *opo*-,

[1]To sustain the chronological order, we cite the earliest peer-reviewed publication available. For the sake of completeness, we also cite later publications on the same work that are more detailed. Thus, in some cases we cite both the conference and journal version of the same work.

and *s*-leaders. As objective functions they consider minimizing the number of bends and the total leader length. While for *opo*-leaders the labels may lie on one, two, or four sides of *R*, the labels for *po*-leaders may lie only on one or on two opposite sides of *R*. Further, Bekos et al. [Bek+06a] consider *opo*-labelings such that labels appear in multiple columns besides *R*. Apart from that, Bekos et al. [Bek+06b, Bek+10b] investigate *opo*-labelings where the sites may *float* within predefined polygons in *R*.

Boundary labeling using *do*-leaders was introduced by Benkert et al. [Ben+07, Ben+09] in 2007. They investigate algorithms minimizing a general badness function on *do*- and *po*-leaders and, furthermore, give more efficient algorithms for the case that the total leader length is minimized. Bekos et al. [Bek+08, Bek+10a] present further optimization algorithms for *do*-leaders and similarly shaped leaders. In 2010 Nöllenburg et al. [NPS10] considered *po*-labelings for a setting that supports interactive zooming and panning and allows stacking of free-floating labels. In 2011 Gemsa et al. [GHN11, GHN15] studied the labeling of panorama images using vertical *s*-leaders. Leaders based on Beziér curves and *s*-leaders are further considered in the context of labeling focus regions by Fink et al. [Fin+12]. Moreover, Huang et al. [HPL14] investigate *opo*-labelings with flexible label positions. Recently, Fink and Suri [FS16] presented dynamic programming approaches for boundary labeling for uniform labels considering the four major leader types.

Boundary labeling is combined in a *mixed model* with internal labels, i.e., labels that are placed next to the sites [LN10, Bek+11, LNS15]. *Many-to-one* boundary labeling is a further variant, where each label may connect to multiple sites [LKY07, LKY08, Lin10, Bek+13]. Finally, boundary labeling is considered in the context of *text annotations* [LWY09, KLW14].

In contrast to the more applied research of the previous section, the above contributions on external label placement mainly focus on simpler abstract models and objectives, for which they mathematically prove quality guarantees. These results can be seen as proofs of concepts that reduce the label placement problem to its mathematical core; yet several of the papers report experimental results and case studies from prototypical implementations. Their abstract view of the problem also prepares the ground for new labeling layouts. Not only *s*-leaders are studied, but a variety of different leader types and their properties are investigated. In total, we listed three papers studying *do*-leaders, nine studying *opo*-leaders, eight studying *po*-leaders, and four papers studying *s*-leaders; see Table 10.1 for a summary. Most of the contributions (15 out of 19) include algorithms optimizing the total leader length.

Table 10.1: Summary of related work broken down into the considered leader types. The considered leader types are marked by $\times$. If a natural extension of a leader type has been investigated, then it is marked by $\star$. TLL: Among others, the authors consider total leader length minimization as objective.

Year	**Reference**	**Leader Type**					**TLL**
		s	*po*	*do*	*opo*	other	
2004	Bekos et al. [Bek+04, Bek+07]	$\times$	$\times$		$\times$		$\times$
2006	Bekos et al. [Bek+06a]				$\times$		
	Bekos et al. [Bek+06b, Bek+10b]				$\times$		$\times$
2007	Benkert et al. [Ben+07, Ben+09]		$\times$	$\times$			$\times$
	Lin et al. [LKY07, LKY08]		$\star$		$\star$		$\times$
2008	Bekos et al. [Bek+08, Bek+10a]			$\times$		$\times$	$\times$
2009	Lin et al. [LWY09]				$\times$	$\times$	$\times$
2010	Lin [Lin10]				$\star$		$\times$
	Löffler, Nöllenburg [LN10]		$\times$				
	Nöllenburg et al. [NPS10]		$\times$				$\times$
2011	Bekos et al. [Bek+11]				$\times$		
	Gemsa et al. [GHN11, GHN15]	$\times$					$\times$
2012	Fink et al. [Fin+12]	$\times$				$\times$	$\times$
2013	Bekos et al. [Bek+13, Bek+15]		$\star$				$\times$
	Kindermann et al. [Kin+13b, Kin+16]		$\times$				$\times$
2014	Huang et al. [HPL14]				$\times$		$\times$
	Kindermann et al. [KLW14]	$\times$	$\times$		$\times$	$\times$	
2015	Löffler et al. [LNS15]			$\times$			$\times$
2016	Fink, Suri [FS16]	$\times$	$\times$	$\times$	$\times$		$\times$
$\sum$		5	9	4	10	4	15

11 On the Readability of Leaders in Boundary Labeling

Abstract. While external labeling has been extensively investigated from a perspective of automatization, the research on its readability has been neglected. In this chapter, we present the first formal user study on the readability of leader types in external labeling. We consider the four most extensively studied leader types with respect to their performance, i.e., whether and how fast a viewer can assign a feature to its label and vice versa. We give a detailed analysis of the results regarding the readability of the four models and discuss their aesthetic qualities based on the users' preference judgments and interviews.

This chapter is based on and partly taken from joint work with Lukas Barth, Andreas Gemsa and Martin Nöllenburg [Bar+15].

11.1 Introduction

Algorithms for external labeling have been extensively investigated both from a practical and a theoretical perspective, as detailed in the previous chapter. Yet, research on the readability of the different labeling models, in particular of the different leader types, has been neglected in literature so far. There exist several user studies on the readability and aesthetics of graph drawings. For example, Ware et al. [War+02] studied how people perceive links in node-links diagrams. However, to the best of our knowledge, there are no user studies[1] on the readability of any of the fundamental boundary labeling models introduced in Chapter 10. In this chapter, we present the first user study on readability aspects of different leader types in boundary labeling. When reading a boundary labeling the viewer typically wants to find for a given site its corresponding label, or vice versa. Hence, a well readable labeling must facilitate this basic two-way task such that it can be performed fast and correctly. We call this the *assignment task*. In this chapter, we investigate the assignment task for the four most established models, namely models using *s*-, *po*-, *opo*- and *do*-leaders, respectively. See Chapter 10 for a detailed discussion on preceding research on these leader types. To keep the number of parameters small, we refrained from considering other types of leaders. We conducted a controlled user study with 31 subjects. Further, we interviewed eight participants about their personal assessment of the leader types. We obtained the following main results.

[1]One exception is the user study by Madsen et al. [Mad+16], but they evaluated the effects of label placement in object vs. image space and update frequency in augmented reality visualizations.

- Type-*opo* leaders lag behind the other leader types in all considered aspects.
- In the assignment task, *do*-, *po*- and *s*-leaders have similar error rates, but *po*-leaders have significantly faster response times than *do*- and *s*-leaders.
- The participants prefer the leader types in the order *do*, *po*, *s* and *opo*.

In the remainder of this chapter, we use the notation for boundary labeling introduced in Chapter 10; in particular we denote the bounding rectangle of the figure by R.

11.2 Research Questions

As argued before, a well readable boundary labeling must allow the viewer to quickly and correctly assign a label to its site and vice versa. More specifically, the leader λ connecting the label with its site must be easily traceable by a human. We hypothesize that both the response time and the error rate of the assignment task significantly depend on other leaders running close to and parallel to λ in the following sense. *The more parallel segments closely surround λ, the more the response time and the error rate of the assignment task increase.*

However, we did not directly investigate this hypothesis, but we derived from it two more concrete hypotheses that are based on the four leader types. These were then investigated in the user study. To that end, we additionally observe, that in medical figures the density of the sites varies. Both may occur, figures containing a *dense set* of sites, where the sites are placed closely to each other, and figures containing a *sparse set* of sites, where the sites are dispersed. We now motivate the hypothesis as follows.

By definition of the models, the number of parallel leader segments in *do*-, *po*- and *opo*-labelings is linear in the number of labels per leader, because each pair of leaders has at least one pair of parallel segments. For *opo*-labelings each pair of leaders even has up to three pairs of parallel segments. Additionally, the spacing of the first orthogonal segments of *opo*-leaders is determined by the y-coordinates of the sites rather than by the (more regularly spaced) y-coordinates of the label ports as in *po*- and *do*-labelings. In contrast, in an *s*-labeling the leaders typically have different slopes, so that (almost) no parallel line segments occur. In fact, it is known that the human eye can distinguish angular differences as small as $10'' \approx 0.003°$ [War12]. Hence, leaders of *do*-, *po*- and *opo*-labelings, in particular for a dense set of sites, are closely surrounded by parallel segments, while *s*-leaders for such a set have very different slopes. We therefore propose the next hypothesis.

(H1) *For instances containing a dense set of sites,*

(a) *the assignment task on s-labelings has a significantly smaller response time and error rate than on do-, po-, and opo-labelings.*

(b) *the assignment task on do- and po-labelings has a significantly smaller response time and error rate than on opo-labelings.*

Considering a sparse set of sites, *do*- and *po*-labelings still have many parallel line segments, but this time they are more dispersed. This is normally not true for *opo*-leaders because the actual routing of these leaders occurs in a thin routing area at the boundary of R. Hence, we propose the next hypothesis.

(H2) *For instances containing a sparse set of sites, the assignment task on opo-labelings has a significantly greater response time and error rate than on do-, po-, and s-labelings.*

In summary, we expect that *opo*-labelings perform worse than the other three, that *do*- and *po*-labelings perform similar, and that *s*-labelings perform best.

11.3 Design of the Experiment

In this section we present the design of our user study. We first describe the tasks that were performed by the participants. Afterwards, we describe in detail which stimuli were presented to the participants, and conclude the section by describing how the stimuli were presented to the participants.

11.3.1 Tasks

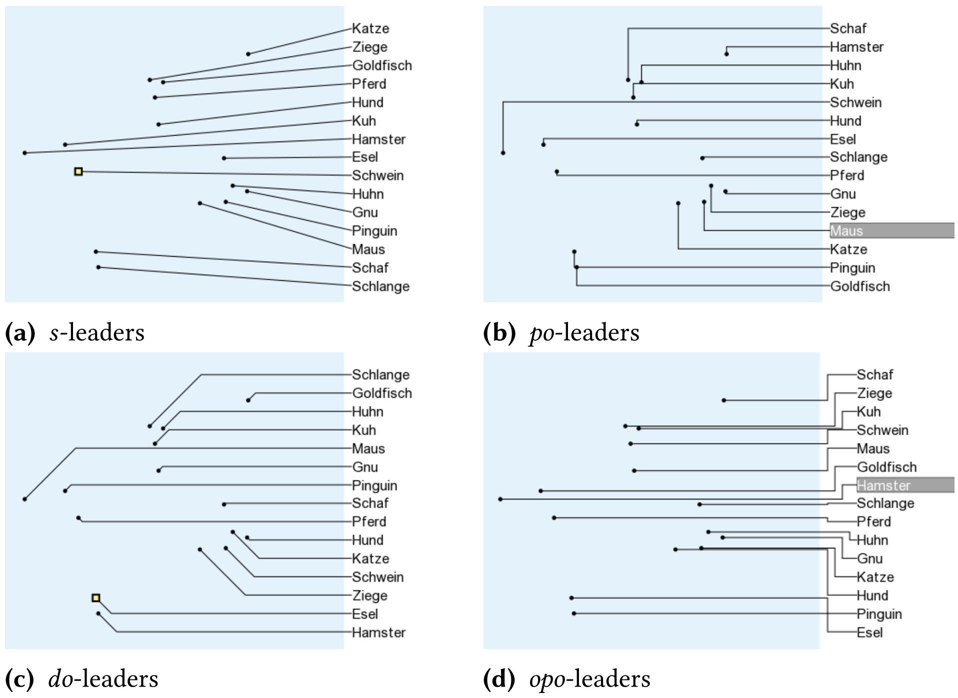
In order to test our hypotheses we presented instances of boundary labeling to the participants and asked them to perform the following two tasks.

1. Label-**S**ite-Assignment (T_S): In an instance containing a highlighted label select the corresponding site.
2. Site-**L**abel-Assignment (T_L): In an instance containing a highlighted site select the corresponding label.

Both tasks are fundamental to reading labeled illustrations and require the subjects to visually follow the selected leader from site to port and from port to site, respectively.

11.3.2 Stimuli

The stimuli are automatically generated boundary labelings, each using the same drawing style. In order to avoid confounding effects between background image and leaders we use a plain light blue background so that the only graphical features affecting the experiment are the sites and their leaders. Sites, leaders and label texts are drawn in the same style and in black color. Highlighted points are drawn as slightly larger yellow-filled squares with black boundary rather than small black disks. Highlighted labels are shown as white text on a dark gray background. This is to minimize the time spent for localizing the relevant features rather than following the leader. Figure 11.1 exemplarily shows four stimuli.



(a) *s*-leaders **(b)** *po*-leaders **(c)** *do*-leaders **(d)** *opo*-leaders

Figure 11.1: Examples of stimuli for both tasks and all four leader types. To avoid learning effects of the participants the order of the animal names are chosen randomly. While in stimuli (a) and (c) sites are highlighted, in stimuli (b) and (d) labels are highlighted.

Choice of Stimuli. For all instances we defined R to be a rectangle of 500×750 pixels. In addition to the four leader types as the main factor of interest, we identified three secondary factors that may have an impact on the resulting labelings. This yields four parameters to classify an instance.

The first parameter is the *number* $\mathcal{N} = \{15, 30\}$ of sites that are contained in the instance. We have chosen 15 sites to obtain small instances and 30 sites to obtain large instances, which are typical numbers, e.g., for medical drawings.

The second parameter is the *distribution* $\mathcal{D} = \{\mathrm{D_U}, \mathrm{D_3}, \mathrm{D_{10}}\}$ that is used for randomly placing the sites in R. We define $\mathrm{D_U}$ to be a uniform distribution. This distribution yields instances whose sites are dispersed in R without having a certain spatial structure. However, considering, e.g., medical drawings, the instances often consist of spatial clusters. We model such a single cluster by a normal distribution. More precisely, we define $\mathrm{D_3}$ and $\mathrm{D_{10}}$ to be normal distributions with mean $\mu = (250, 375)$ at the center of R, and variance $\sigma^2 = 3000$ and $\sigma^2 = 10000$ in both dimensions, respectively. Hence, $\mathrm{D_3}$ yields instances consisting of a dense cluster of sites, while $\mathrm{D_{10}}$ yields instances consisting of a sparse cluster of sites. In order to avoid cluttered sets of sites and degenerated instances, where sites lie too close to the boundary of R, we rejected instances where any two sites have less than 10 pixels distance or where a site has less than 30 pixels distance to the boundary of R.

The third parameter is the applied *leader type* $\mathcal{T} = \{do, opo, po, s\}$ as defined in Chapter 10. Finally, the fourth parameter $\mathcal{R} = \{0.3, 0.6, 0.9\}$ can be seen as a difficulty level and specifies which leader of the instance should be selected for the tasks $\mathrm{T_S}$ and $\mathrm{T_L}$. This is accomplished by scoring each leader with respect to how much ink is close to it in the drawing. More specifically, ranking a leader λ_i is done as follows: For every other leader λ_j, points are linearly sampled on λ_j with one pixel distance from each other. For each such point, the minimum distance d to λ_i is computed. Then, every sample point contributes $\frac{1}{d^2}$ to the *ink score* of λ_i. The parameter $r \in \mathcal{R}$ then selects the leader λ whose ink score is the r-quantile among the ink scores of all leaders in the instance. Thus the parameter $\mathcal{R}$ lets us control the relative difficulty of the chosen leader. Later on in our analysis we do not break down the results into the introduced difficulty levels. Their purpose is to create stimuli of varying difficulty. We do, however, confirm in Section 11.4.3 that those levels indeed create instances of varying difficulty.

Thus, the parameter space $\mathcal{N} \times \mathcal{D} \times \mathcal{T} \times \mathcal{R}$ lets us cover a large variety of different sample instances that are representative of real-world instances. Figure 11.4 and Figure 11.5 in Section 11.7 exemplarily show stimuli for different choices of the presented parameters.

Generation of Stimuli. For each of the $2 \cdot 3 \cdot 4 \cdot 3 = 72$ possible choices of parameters $(n, d, t, r) \in \mathcal{N} \times \mathcal{D} \times \mathcal{T} \times \mathcal{R}$ we have generated two stimuli I_1 and I_2, one for each task. To that end, we used the property that a leader of any of the four types is uniquely determined by the position of its port and site. Using integer linear programming (ILP) as an exact optimization method, we computed a length-minimal labeling of the chosen leader type such that the labels are placed to the right side of R using one of 150 equally spaced ports each; a formal description of the ILP is found at the end of this section. The ILP ensures that the subset of chosen ports does not create overlapping labels and that no two leaders cross. The sample of 150 available ports simulates labels that may be placed anywhere on the right side of R. The sample is dense enough so that a further refinement has no relevant effect on the visual appearance of the stimuli and sparse enough as to allow fast solution times of the ILP models. In each instance the text of each label is randomly chosen from a pre-defined set of German animal names. For *opo*-labelings, the required track routing area and the routing of the leaders is such that the p-segments of any two leaders have horizontal distance of at least 10 pixels from each other.

It will occur in the instances that leaders lie closely together, e.g., see Figure 11.1(d). However, we do not enforce minimum spacing between leaders because neither any of the studied models nor any of the discussed algorithms enforce minimum spacing explicitly. In fact, a fixed minimum leader spacing may even lead to infeasible instances for certain leader types. Rather, one can see the observed closeness of leaders as a predisposition of the respective leader type.

We now describe the ILP for generating labelings formally. To that end assume that we are given a certain leader type $t \in \mathcal{T}$, the set S of sites and the set P of ports. For each pair $(s, p) \in S \times P$ we generate the *candidate* $c_{s,p} = (\ell, \lambda)$, where ℓ is a label placed at p (i.e., the midpoint of ℓ's left side is located at p), and, furthermore, λ is a leader of type t connecting s with p. We say two candidates $c = (\ell, \lambda)$ and $c' = (\ell', \lambda')$ *intersect*, if ℓ and ℓ' or λ and λ' intersect. We introduce for each candidate $c_{s,p}$ a binary variable $x_{s,p} \in \{0, 1\}$. We interpret $x_{s,p}$ such that the label ℓ of s is placed at p if and only if $x_{s,p} = 1$. We introduce the following constraints.

$$\sum_{p \in P} x_{s,p} = 1 \text{ for all } s \in S \tag{11.1}$$

$$x_{s,p} + x_{t,q} \leq 1 \text{ for all } s, t \in S \text{ and } p, q \in P \\ \text{such that } c_{s,p} \text{ and } c_{t,q} \text{ intersect} \tag{11.2}$$

Subject to those constraints, we minimize the objective

$$\sum_{s \in S, p \in P} x_{s,p} \cdot \text{length}(c_{s,p}), \tag{11.3}$$

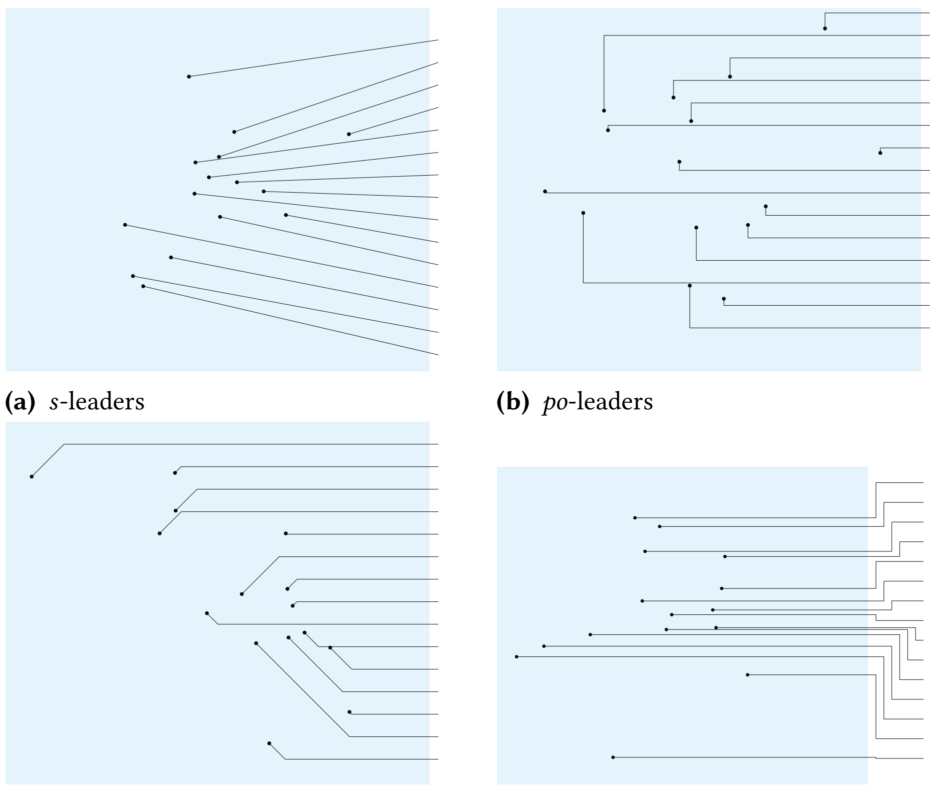
where length$(c_{s,p})$ is the length of the leader of the candidate $c_{s,p}$. Constraint (11.1) ensures that for each site exactly one port p is selected, where the label is placed. Constraint (11.2) enforces that the placed labels as well as their leaders do not intersect in any way. Finally, the objective minimizes the total length of the leaders.

Note that in case of *opo*-leaders we create the candidates such that the vertical segments of the leaders all lie on the right side of R. For the pairwise intersection tests, overlaps of vertical segments are not taken into account. In a post-processing step the vertical segments are horizontally arranged in the track routing area as described above.

Finding an optimal solution for an ILP formulation is NP-hard in general. However, it turns out that in practice we can apply specialized solvers to find optimal solutions for reasonably sized instances in acceptable time. Hence, this ILP-based method provides a simple and generic way to produce our stimuli, without the need for implementing specialized algorithms for each type.

11.3.3 Procedure

The study was run as a within-subject experiment. Four experimental sessions were held in the computer lab of our institute at KIT at controlled lighting with 12 identical machines and screens using a digital questionnaire in German language. After agreeing



(a) *s*-leaders **(b)** *po*-leaders

(c) *do*-leaders **(d)** *opo*-leaders

Figure 11.2: Instances presented as examples next to the personal preference questions.

to a consent form, each participant first completed a tutorial explaining to him or her the tasks T_S and T_L on four instances, each containing one of the four labeling types. Participants were instructed to answer the questions as quickly and as accurately as possible. Afterwards, the actual study started presenting the 144 stimuli to the participant one at a time. Each stimulus was revealed to the participant, after he or she clicked a button in the center of the screen using the mouse. Hence, at the beginning of each task the mouse pointer was always located at the same position. Then he or she performed the task by selecting a label or site using the mouse.

The stimuli were divided into 12 blocks consisting of 12 stimuli each. Each block either contained stimuli only for T_S or only for T_L. For each participant the stimuli were in random order, but in alternating blocks, i.e., after completing a block for T_S, a block for T_L was presented, and vice versa. Between two successive blocks a pause screen stated the task for the next block and participants were asked to take a break of at least 15 seconds before continuing.

Especially for professional printings, e.g. for atlases of human anatomy, not only the figure's readability, but also its aesthetics is seen to be of great importance. Further, assigning a label to its site (or vice versa), the viewer should be able to assess whether he or she has done this correctly. We therefore asked all participants about their personal assessment of the aesthetics and readability of the leader types after completing the 144 performance trials. We presented the same four selected instances of the four leader types to each participant. To that end, we selected an instance for each leader type $t \in \mathcal{T}$ based on the 144 instances generated for the tasks $\mathrm{T_S}$ and $\mathrm{T_L}$. We scored each instance by the sum of its leaders' ink scores. Among all instances with leader type $t \in \mathcal{T}$ and 15 sites, we selected the median instance I with respect to the instance scores of that subset. Hence, for each type of leader we obtain a moderate instance with respect to our difficulty measure; see Figure 11.2. Each participant was asked to rate the different leader types using German school grades on a scale from 1 (excellent) to 6 (insufficient), where grades 5 and 6 are both fail-grades, by answering the following questions.

Q1. How do you rate the appearance of the leaders?
Q2. For a highlighted site, how easy is it for you to find the corresponding label?
Q3. For a highlighted label, how easy is it for you to find the corresponding site?

We further conducted short interviews with eight participants after the experiment, in which they justified their grading.

11.4 Results

In total 31 students of computer science in the age between 20 and 30 years completed the experiment, six of them were female and 25 were male. We also asked whether they have substantial knowledge about labeling figures and maps, which was affirmed by only two participants. Hence most participants of our study were in fact non-expert users without much background knowledge that would bias their judgments.

11.4.1 Performance Analysis

For each of the 144 trials we recorded both the response time and the correctness of the answer,[2] which allows for analyzing two separate quantitative performance measures. Response times were measured from the time a stimulus was revealed until the participant clicked to give the answer. Response times are normalized per participant by his/her median response time to compensate for different reaction times among participants. We split the data into four groups by leader type, and refer to the groups by $\mathcal{S}$, $\mathcal{PO}$, $\mathcal{DO}$ and $\mathcal{OPO}$, respectively.

[2]Data available at `http://i11www.iti.kit.edu/projects/bl-userstudy`

Table 11.1: Mean normalized response times for overall processed tasks and for correctly processed tasks as well as mean success rates. The times and rates are broken down into dense, sparse and uniform sets of sites, into large and small instances, as well as the tasks T_S and T_L.

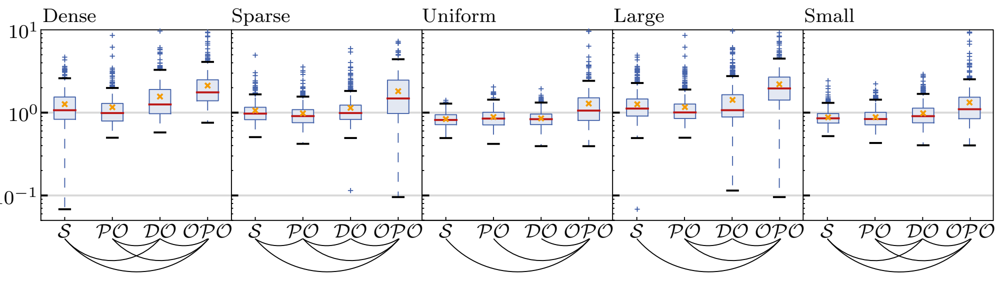
	Overall processed tasks				*Correctly processed tasks*			
	$\mathcal{S}$	$\mathcal{PO}$	$\mathcal{DO}$	$\mathcal{OPO}$	$\mathcal{S}$	$\mathcal{PO}$	$\mathcal{DO}$	$\mathcal{OPO}$
Dense	1.266	1.161	1.564	2.122	1.262	1.142	1.552	2.020
Sparse	1.063	0.981	1.143	1.813	1.065	0.980	1.132	1.667
Uniform	0.836	0.885	0.852	1.287	0.837	0.894	0.855	1.231
Large	1.262	1.167	1.425	2.201	1.262	1.158	1.405	2.083
Small	0.848	0.852	0.949	1.281	0.850	0.854	0.948	1.239
T_S	1.074	1.083	1.276	1.748	1.072	1.086	1.266	1.602
T_L	1.037	0.936	1.098	1.743	1.032	0.922	1,081	1.648

We applied repeated-measures Friedman tests with post-hoc Dunn-Bonferroni pairwise comparisons in SPSS[3] between the four groups to find significant differences in the performance data at a significance level of $p = 0.05$. We chose a non-parametric test since our data is not normally distributed. We report the detailed test results in Table 11.5 in Section 11.7 and summarize the main findings in the following paragraphs.

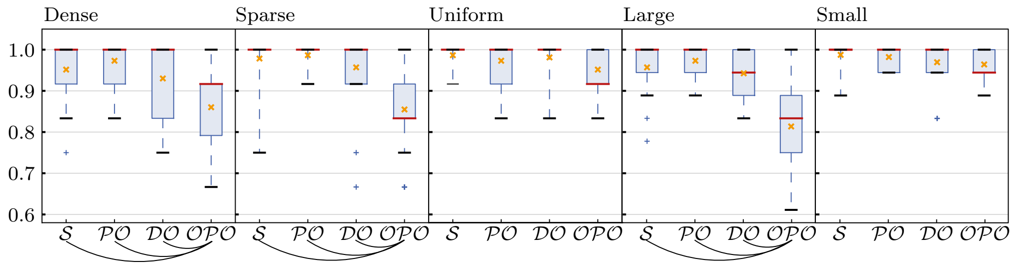
Response Times. The first three plots of Figure 11.3(a) show the normalized response times broken down into the three considered distributions D_3, D_{10} and D_U, which yield *dense*, *sparse* and *uniform* sets of sites; the corresponding mean times are found in Table 11.1. Further plots for both normalized and absolute response times are found in Figure 11.6, Figure 11.7 and Figure 11.8 in Section 11.7. We obtained the following results. Among all leader types, *opo*-leaders have the highest response time. In particular for dense and sparse sets of sites the mean response time is up to a factor 1.8 worse than for the others. For uniform sets we obtain a factor of up to 1.5. Further, for any distribution the measured differences with respect to *opo*-leaders are significant.

Comparing the response times of the remaining leader types, we obtain the order $po < s < do$ with respect to increasing mean response time. For uniform sets we did not measure any pairwise significant difference between *do*, *po* and *s* leaders. However, for dense and sparse sets we obtained the significant differences as shown in Figure 11.3(a). We emphasize that for *po*- and *s*-leaders significant differences are measured for sparse, but not for dense sets of sites. In contrast *do*- and *s*-leaders have significant differences for dense sets, but not for sparse sets. Further, *po*- and

[3] `http://www-01.ibm.com/software/analytics/spss/`



(a) Normalized response times (logarithmic scale). Smaller values are better than higher values.



(b) Success rates. Higher values are better than smaller values.

Figure 11.3: Performance results broken down to dense, sparse and uniform sets as well as to large instances (30 sites). Mean values are indicated by 'x'. Arcs at the bottom show significant differences that were found ($p = 0.05$).

do-leaders have significant differences in both dense and sparse sets. Altogether, this justifies the ranking $po < s < do$ w.r.t. increasing mean response time.

Comparing the instances in terms of the two tasks T_S and T_L, the mean response time of T_L is slightly faster than that of T_S. A possible explanation is that clicking a site requires more precise mouse movements than clicking a label. Filtering out incorrectly processed tasks does not change the mean response time much and similar results are obtained; see Table 11.1.

The two last plots of Figure 11.3(a) show the mean response times of *large instances* (any instance with 30 sites and dense, sparse or uniform distribution) and of *small instances* (any instance with 15 sites and dense, sparse or uniform distribution), respectively. While for large instances the mean response times are similar to those of dense sets, the mean response times of small instances are similar to those of uniform sets. In both cases, the response times for *opo*-leaders are significantly higher than for any other leader type. Hence, likely to sparse and uniform instances, *opo*-leaders lags behind the other leader types.

$\mathcal{S}$	$\mathcal{PO}$	$\mathcal{DO}$	$\mathcal{OPO}$
0.952	0.973	0.930	0.860
0.978	0.987	0.957	0.855
0.987	0.973	0.981	0.952
0.945	0.973	0.943	0.814
0.988	0.982	0.970	0.964
0.982	0.991	0.959	0.886
0.962	0.964	0.953	0.892

Table 11.2: Mean success rates broken down into the different leader types.

Accuracy. We computed for each leader type and each participant the proportion of instances of that type that the participant solved correctly; see Figure 11.3(b) and Table 11.2. For dense and sparse sets of sites we observe that $\mathcal{OPO}$ has success rates around 86%, while the other groups have success rates greater than 93%. In particular the differences between success rates of *opo*-leaders and the remaining types are up to 11% and 13% for dense and sparse sets, respectively. Any of these differences is significant, while between $\mathcal{S}$, $\mathcal{PO}$ and $\mathcal{DO}$ no significant accuracy differences were measured. For uniform sets of sites, on the other hand, no significant differences were measured and any group has a success rate greater than 95%. Hence, it appears that uniform sets of sites produce easily readable labelings with any leader type – unlike dense and sparse instances.

Considering large and small instances separately, the group $\mathcal{OPO}$ has an inferior success rate (81%), while the other groups remain almost unchanged (> 93%), which yields for $\mathcal{PO}$ and $\mathcal{OPO}$ a difference of 16%. For small instances no significant differences were measured. Comparing the instances by tasks T_S and T_L, the success rates are very similar, but for T_S it is slightly better than that for T_L except for $\mathcal{OPO}$.

11.4.2 Preference Data

Table 11.3 shows the average grades given by the participants with respect to the three questions Q1–Q3. Concerning the general aesthetic appeal (question Q1) leaders of type *do* received the best grades (1.8), followed by *po*-leaders (grade 2.3). The participants did not particularly like the appearance of *s*-leaders (grade 3.3) and generally disliked *opo*-leaders (grade 4.6). Table 11.4 lists the detailed percentages of participants who graded a particular leader type better, equally, or worse than another type.

In addition to the general impression from the average grades it is worth mentioning that between the two most preferred leader types *do* and *po* 48.4% preferred *do* over *po* and 38.7% gave the same grades to both leader types. Compared to the *s*-leaders, a great majority (> 80%) strictly prefers both *do*- and *po*-leaders. In the interviews seven out of eight participants stated that *opo*-leaders are "confusing, because leaders closely pass by each other". They disliked the long parallel segments of *opo*-leaders.

	s	*po*	*do*	*opo*
Q1	3.3	2.3	1.8	4.6
Q2	2.4	2.1	2.0	4.6
Q3	2.4	2.3	1.7	4.3

Table 11.3: Average grades given by the participants with respect to questions Q1–Q3. Smaller values are better than higher values.

Further, some participants remarked that *opo*-leaders “consist of too many bends”. For six participants *s*-leaders were “chaotic and unstructured”, unlike *do*- and *po*-leaders. Five participants said that they liked the flat bend of *do*-leaders more than the sharp bend of *po*-leaders. One participant stated that “*po*-leaders seem to be more *abstract* than *do*-leaders”. Further, it was said that “the ratio of the segments’ lengths is less balanced for *po*- than *do*-leaders.”

For question Q2 (task T_L, site-to-label) *do*- and *po*-leaders were ranked best (see Table 11.3), followed by *s*-leaders and more than two grades behind by *opo*-leaders, whereas for question Q3 (task T_S, label-to-site) *do*-leaders are further ahead of *po*- and *s*-leaders, both of which received similar grades, and are again about two grades ahead of *opo*-leaders. For questions Q2 and Q3 the most striking observation is that type-*s* leaders received much better results (almost a full grade point better) than for Q1. This is in strong contrast to the other three leader types, which received grades in roughly the same range for all three questions. This indicates that the participants perceived straight leaders as being well readable during the experiment, but still not producing very appealing labelings.

In the interviews participants stated that “*opo*-leaders are hard to read because of leaders lying close to each other.” They negatively observed that *opo*-leaders “may not be clearly distinguished”, but assessed the “simple shape of *s*-leaders to be easily legible.” Further, they positively noted that “the distances between *do*-leaders seem to be greater than for other types” and that “*po*-leaders are easier to follow than other types”.

It is remarkable that according to their subjective impressions, participants rated *do*-leaders best, while they ranked third in our performance test. We conjecture that the participants overestimate the performance of *do*-leaders, because they like their aesthetics. For *s*-leaders the reverse is true. In contrast, the self-assessment on *po*- and *opo*-leaders corresponds more closely to the results of our performance test.

In summary, *do*-leaders obtained the best subjective ratings. The regularly shaped *po*- and *do*-leaders both scored better than the irregular and less shape-restricted *s*-leaders. For any of the three questions *opo*-leaders were rated a lot worse than the others, which is, according to the interviews, mostly due to the frequent occurrence of many nearby leaders running closely together. This is an inherent effect of their definition, since *opo*-leaders do not get vertically separated until the track-routing area, whereas all other leader types separate immediately at the sites.

Table 11.4: Statistics for questions Q1–Q3. The percentage of participants that graded a leader type better (<), equally (=) or worse (>) than another. Majorities are highlighted in bold.

	do<opo	*do=opo*	*do>opo*	*do<po*	*do=po*	*do>po*	*do<s*	*do=s*	*do>s*
Q1	**100**	0	0	**48.4**	38.7	12.9	**90.3**	3.2	6.5
Q2	**93.5**	3.2	3.2	32.3	**41.9**	25.8	**48.4**	19.4	32.3
Q3	**93.5**	6.5	0	**54.8**	35.5	9.7	**48.4**	35.5	16.1
	po<opo	*po=opo*	*po>opo*	*po<s*	*po=s*	*po>s*	*s<opo*	*s=opo*	*s>opo*
Q1	**100**	0	0	**80.6**	3.2	16.1	**80.6**	6.5	12.9
Q2	**96.8**	3.2	0	35.5	25.8	**38.7**	**83.9**	9.7	6.5
Q3	**93.5**	3.2	3.2	**41.9**	16.1	**41.9**	**93.5**	3.2	3.2

11.4.3 On the Choice of the Difficulty Levels

In this section, we shortly discuss the choice of the ink-score-based difficulty levels $\mathcal{R} = \{0.3, 0.6, 0.9\}$ that we used for selecting the highlighted labels and sites for the experimental tasks. We want to examine whether they are a reasonable indicator for assessing the difficulty of the stimuli. In the remainder of this subsection we identify the difficulty levels 0.3, 0.6, 0.9 with *easy*, *medium* and *hard*, respectively.

For the analysis we grouped the stimuli so that they only differ in their difficulty level, i.e., all stimuli in the same group have the same number of sites, the same distribution, the same leader type, and they belong to the same task, namely either task $\mathrm{T_S}$ or $\mathrm{T_L}$. This yields $2 \cdot 3 \cdot 4 \cdot 2 = 48$ groups, and we obtain $31 \cdot 48 = 1488$ samples taking the 31 participants into account.

Response Time. For each sample and each difficulty level $r \in \mathcal{R}$ we denote the response time of the corresponding participant by t_r. Thus, for each sample we obtain the three pairs $(t_{0.3}, 0.3)$, $(t_{0.6}, 0.6)$, $(t_{0.9}, 0.9)$. Sorting those three pairs by their response time in increasing order, we obtain a ranking of the observed difficulty levels in terms of the required response times.

For about 36.4% of the samples this ranking coincides with the desired ordering *easy*, *medium*, *hard*. In comparison, assuming a uniform distribution on the orderings, we could only expect 16.6% of the samples having that particular ordering. Further, for over 78.8% of the ranked samples *easy* comes before *hard*. Under a uniform distribution we could expect only 50%.

Accuracy. For each sample and each difficulty level $r \in \mathcal{R}$ let $s_r \in \{0, 1\}$ indicate whether the participant has correctly solved the according task. More precisely, if $s_r = 1$ then the participant has solved the task correctly; otherwise $s_r = 0$. Over 93.2% of the samples preserved the ordering $s_{0.3} \geq s_{0.6} \geq s_{0.9}$, which indicates that an instance with higher difficulty level is at least as difficult as an instance with lower difficulty level. Breaking down the samples into the four leader types, between 87.6%

(*opo*-leaders) and 96.5% (*po*-leaders) of the samples preserve that ordering. Hence, for all leader types we obtain high accordance with the difficulty levels.

These findings suggest that there is indeed a correlation between the considered ink-score-based difficulty levels and the actual difficulty of the instances in terms of response times and accuracy. We conclude that the introduced difficulty levels provide an adequate method to create stimuli of varying difficulty.

11.5 Discussion

In Section 11.2 we hypothesized that labelings with many parallel leaders lying close to each other have a significant negative effect on response times and accuracy. Our results from Section 11.4.1 indeed support hypotheses (H1b) and (H2), which said that the assignment task has a significantly smaller response time and error rate for *do*- and *po*-labelings than for *opo*-labelings in dense (H1b) and also sparse sets of sites (H2). Hypothesis (H2) was claimed to also hold for *s*-labelings versus *opo*-labelings, which is confirmed by the experiment as well. While greater response times may still be acceptable in some cases, the significantly lower accuracy clearly restricts the usability of *opo*-leaders. Only for small numbers of sites and uniform distributions *opo*-leaders have comparable success rates to the other leader types. This judgment is strengthened further by the preference ratings. On average the participants graded *opo*-leaders between 4 (sufficient) and 5 (poor) in all concerns. The main reason given in the interviews was that *opo*-labelings are confusing due to many leaders closely passing by each other.

However, our results falsified hypothesis (H1a), which claimed that for dense instances type-*s* leaders perform significantly better than the other three leader types. Rather we gained unexpected insights into the readability of boundary labeling. While we had expected that due to their simple shape and easily distinguishable slopes *s*-leaders will perform better than all other types of leaders, we could not measure significant differences between *po*-leaders and *s*-leaders. Interestingly, on average, the participants graded *po*-leaders better than *s*-leaders in all examined concerns, in particular with respect to their aesthetics (Q1). This is emphasized by the interview statements given by the participants that *po*-labelings appear structured while *s*-labelings were perceived as chaotic. Comparing *do*- and *s*-leaders we measured some evidence for (H1a), namely that the assignment task has significantly smaller response times for *s*- than for *do*-leaders. However, the success rates did not differ significantly.

We summarize our main findings regarding the four leader types as follows:

(1) *do*-leaders perform best in the preference rankings, but concerning the assignment tasks they perform slightly worse than *po*- and *s*-leaders.

(2) *opo*-leaders perform worst, both in the assignment tasks and the preference rank-

ings. They are applicable only for small instances or for uniformly distributed sites.

(3) *po*-leaders perform best in the assignment tasks, and received good grades in the preference rankings.

(4) *s*-leaders perform well in the assignment tasks, but not in the preference rankings. The participants disliked their unstructured appearance.

11.6 Conclusions

As a consequence of our experiment we can generally recommend *po*-leaders as the best compromise between measured task performance and subjective preference ratings. For aesthetic reasons, it may also be advisable to use *do*-leaders instead as they have only slightly lower readability scores but are considered the most appealing leader type.

To obtain reliable results and exclude confounding factors as much as possible, we have considered a rather clean and simple boundary labeling model omitting any background image and complex image contour. This design decision ensures that we have actually measured the readability of the leader types without risking the influence of unintentional side-effects, e.g., side-effects created by the choice of the specific background image. These fundamental results can be used as point of departure for more sophisticated user studies on the readability of more complex boundary labeling models specifically, and external labeling models in general. For example, in a subsequent user study, one could systematically investigate the influence of background images and more general image contours for the placement of labels.

Moreover, an interesting question is why type-*s* leaders (which showed good task performance) are most frequently used in practice in actual visualization systems or by professional graphic designers, e.g., in anatomical drawings, although they were not perceived as aesthetically pleasing in our experiment. Conversely, type-*po* leaders are much less frequently used in practice despite their better aesthetic ratings and better or equal task performance. In fact, straight-line connections between two points actually yield the shortest possible leaders, do not require an additional bend point and thus might appear as the most natural solution. One explanation for our diverging results may be that the experiment judged all leader types on an empty, rectangular background, where the leaders receive the entire visual attention of a viewer. In reality, the labeled figure itself is the main visual element and the leaders should be as unobtrusive as possible and not interfere with the figure. Do *s*-leaders produce less interference? Does their perceived "chaotic appearance" decrease in the presence of an actual image? It would be necessary to conduct further experiments to assess the influence and interplay of image and leaders on more complex readability tasks. Yet, our results show that if there is a good contrast between background and leader as in our experiment, the additional bend of *po*-leaders has no negative effect on task performance and the more structured leader shape is positively perceived.

Another interesting follow-up question is whether the chosen objective function to minimize the total leader length (or ink) produces actually the most aesthetic and most readable labelings. Despite being the predominant objective function in the (algorithmic) literature on boundary labeling, simply minimizing the total leader length most certainly does not capture all relevant quality criteria, in particular if the context of the labeled image is taken into account. Yet, it would be surprising if allowing slightly longer leaders had a strong effect on the relative performance of the four leader types in our experimental setup.

Finally, interactive visualization models offer ample space for research on the readability of different labeling styles. With the increasing importance of digital devices, interactive applications play more and more a major role in information visualization. In particular, this creates the possibility to interactively emphasize the correspondence between image features and their labels by highlighting them on demand, e.g., when hovering over a label, the corresponding site and leader are highlighted. It is to be expected that this significantly increases the accuracy of the user, but it may take more time to select a site or label. However, a detailed user study on interaction techniques for external labeling remains future work. Nonetheless, in traditional media (e.g., books, magazines, television) non-interactive images remain important, and the appropriate choice of the leader type is essential.

11.7 Additional Plots and Tables

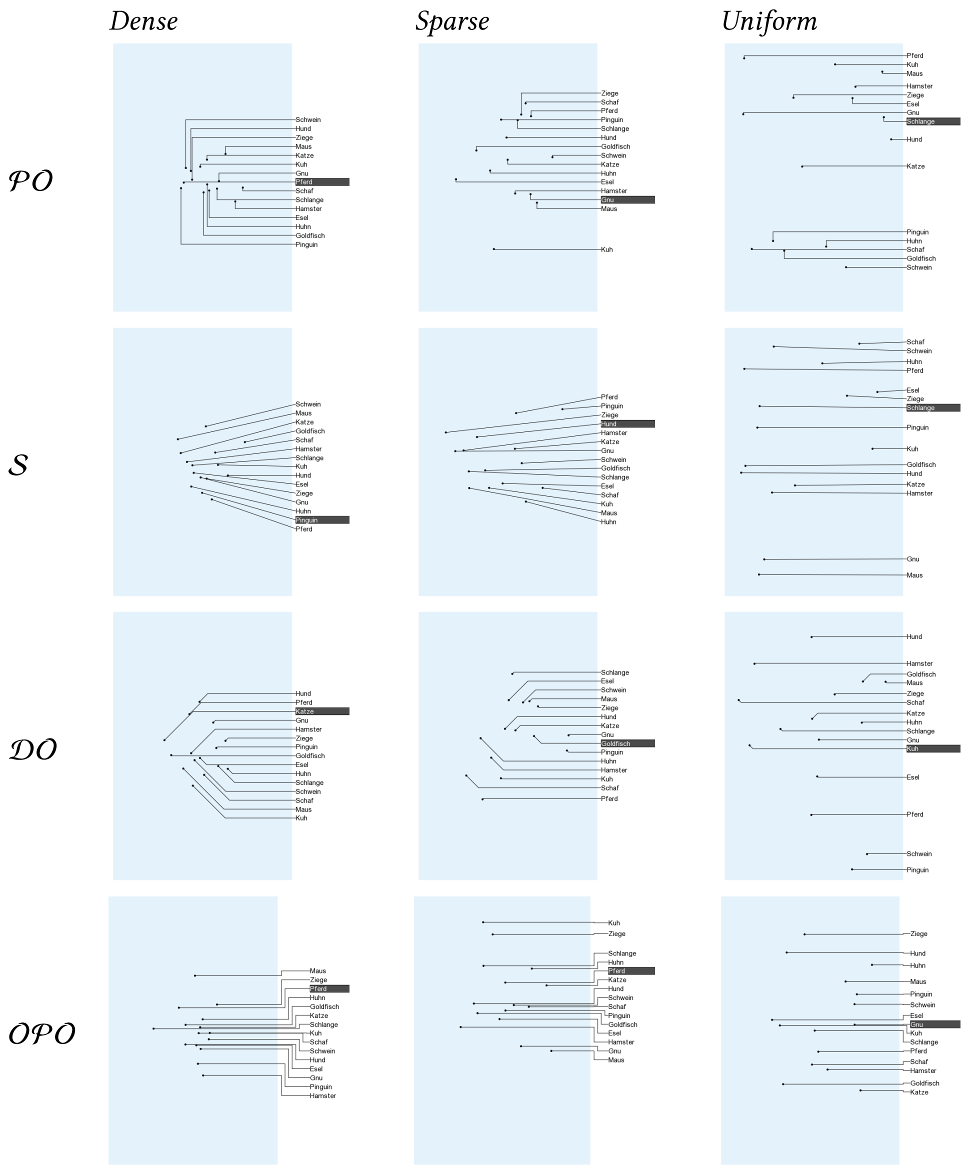


Figure 11.4: Example stimuli with 15 sites (small), one for each site distribution and for each leader type. Due to formatting the rectangles enclosing the sites may not have same sizes. In the digital questionnaire they had the same size.

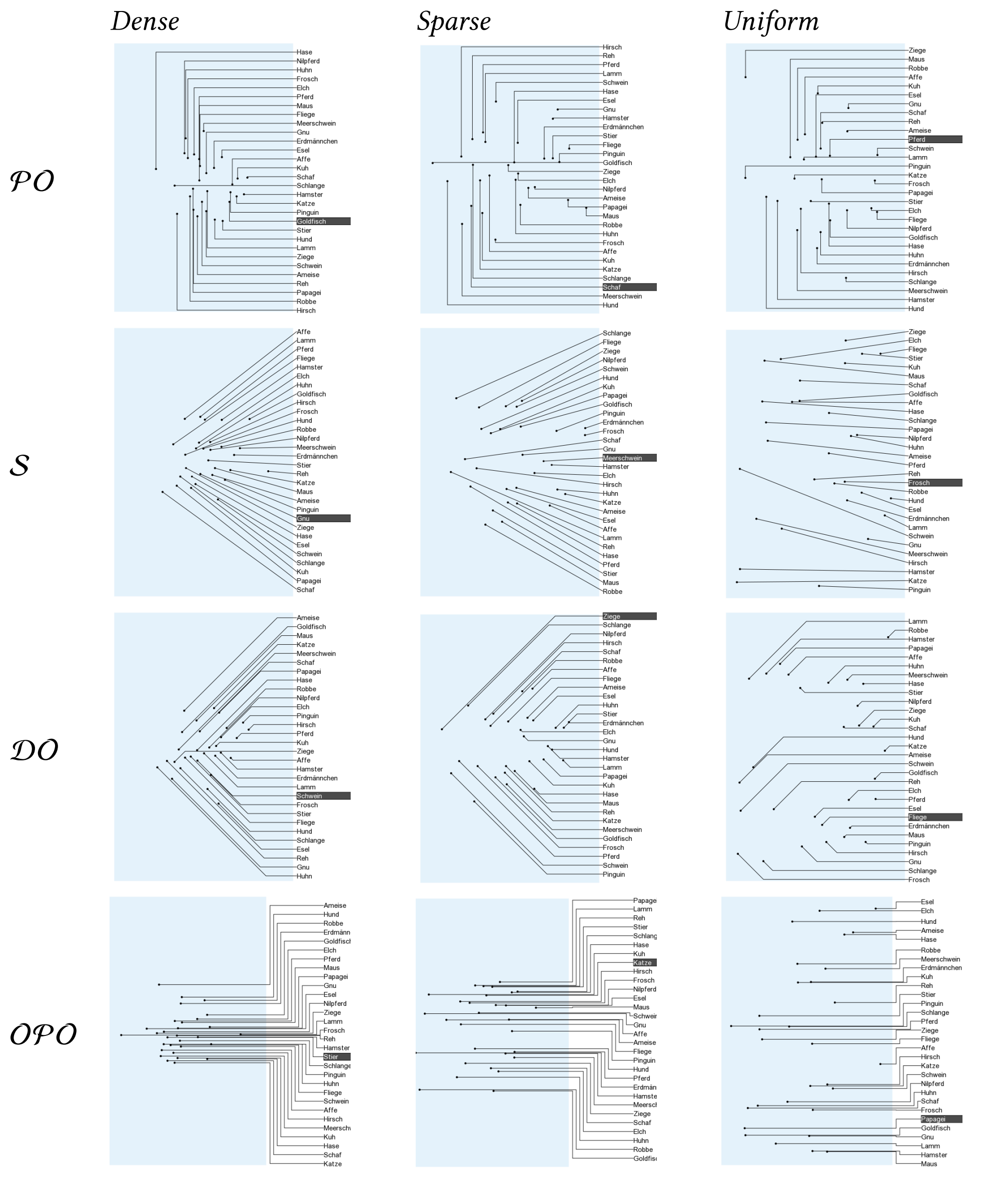


Figure 11.5: Example stimuli with 30 sites (large), one for each site distribution and for each leader type. Due to formatting the rectangles enclosing the sites may not have same sizes. In the digital questionnaire they had the same size.

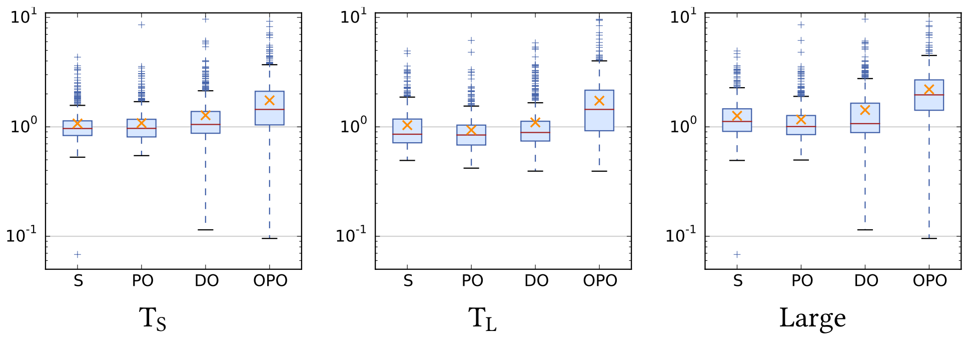
Table 11.5: Results of the Dunn-Borferroni test on (a) the response times and (b) the success rates of the respective pairs of groups. The values estimate the likelihood that both respective sample groups are from the same population. A value < 0.05 is treated as statistically significant difference (marked green). OPT: all processed tasks. CPT: correctly processed tasks only. T_S/T_L: Restricted to instances of task T_S/T_L. *Dense*/*Sparse*/*Uniform*: Restricted to instances of distribution dense/sparse/uniform. *Large*/*Small*: Restricted to instances containing 30 and 15 sites, respectively.

	$\mathcal{PO}$-$\mathcal{S}$	$\mathcal{S}$-$\mathcal{DO}$	$\mathcal{PO}$-$\mathcal{DO}$	$\mathcal{DO}$-$\mathcal{OPO}$	$\mathcal{S}$-$\mathcal{OPO}$	$\mathcal{PO}$-$\mathcal{OPO}$
OPT:T_S	1.0	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
OPT:T_L	0.010	0.382	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
OPT:*Dense*	0.415	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
OPT:*Sparse*	0.001	1.0	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
OPT:*Uniform*	0.263	1.0	0.129	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
OPT:*Large*	0.335	0.098	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
OPT:*Small*	1.0	0.001	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
CPT:T_S	1.0	0.001	0.001	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
CPT:T_L	0.281	0.174	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
CPT:*Dense*	1.0	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
CPT:*Sparse*	0.002	1.0	0.003	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
CPT:*Uniform*	0.125	1.0	0.135	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
CPT:*Large*	1.0	0.221	0.013	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
CPT:*Small*	1.0	0.001	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$

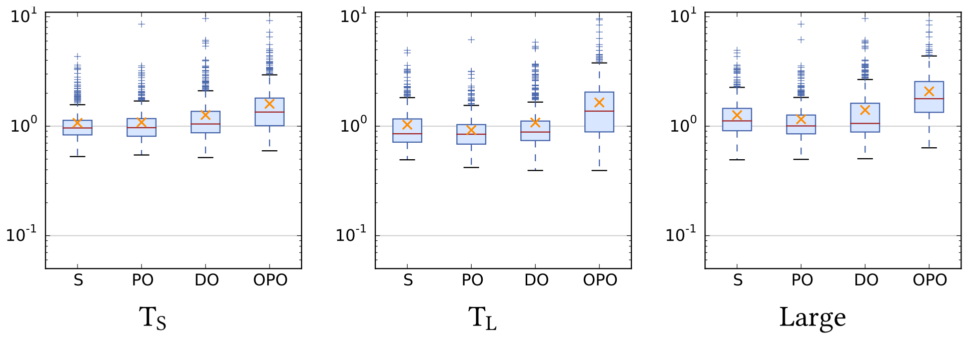
(a) Response times.

	$\mathcal{PO}$-$\mathcal{S}$	$\mathcal{S}$-$\mathcal{DO}$	$\mathcal{PO}$-$\mathcal{DO}$	$\mathcal{DO}$-$\mathcal{OPO}$	$\mathcal{S}$-$\mathcal{OPO}$	$\mathcal{PO}$-$\mathcal{OPO}$
T_S	1.0	1.0	0.460	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
T_L	1.0	1.0	1.0	0.001	$< 10^{-3}$	$< 10^{-3}$
Dense	1.0	1.0	0.330	0.019	0.001	$< 10^{-3}$
Sparse	1.0	1.0	1.0	0.001	$< 10^{-3}$	$< 10^{-3}$
Uniform	1.0	1.0	1.0	0.262	0.125	1.0
Large	1.0	1.0	0.922	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
Small	1.0	0.764	1.0	1.0	0.055	0.262

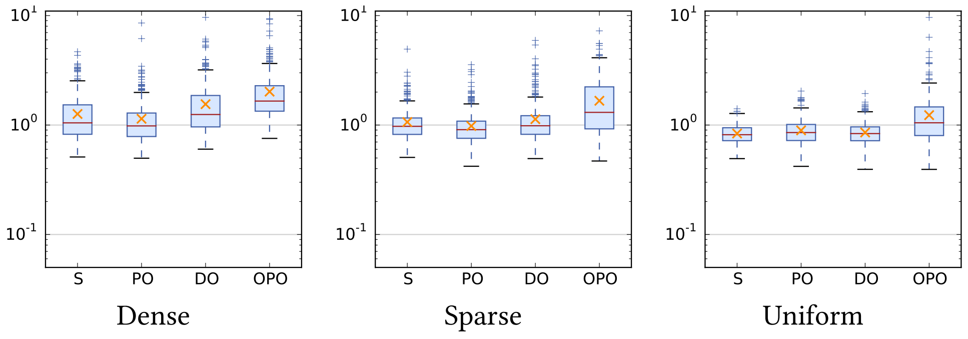
(b) Success rates.



(a) Response times over all tasks (OPT).

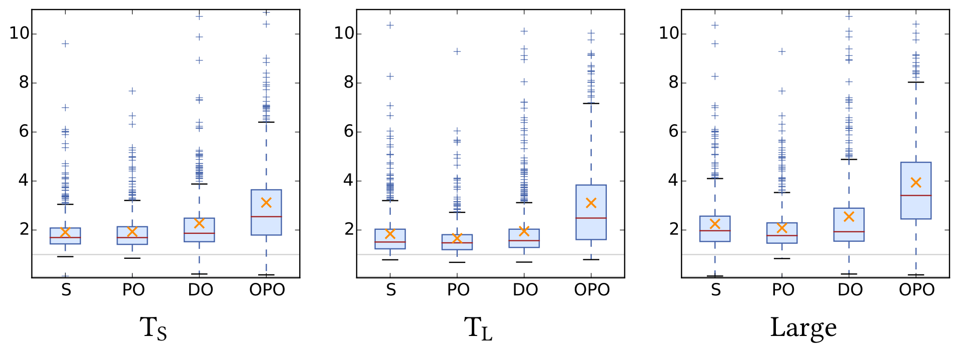


(b) Response times over all correctly processed tasks (CPT).

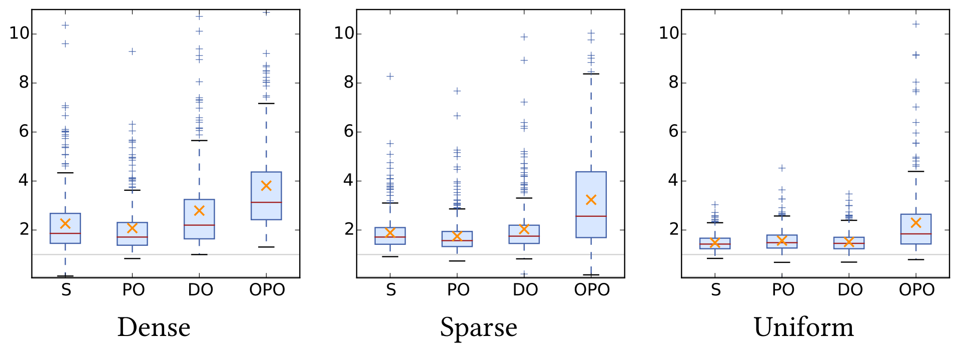


(c) Response times over all correctly processed tasks (CPT).

Figure 11.6: Normalized response times (on log-scale) broken down to different parameters. Mean values are indicated by a bold 'x'. The corresponding significances are found in Table 11.5. Smaller values are better than higher values.

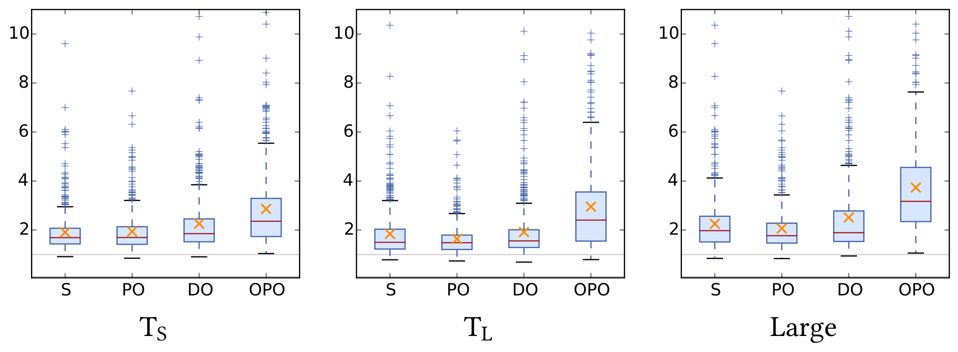


(a) Response times in seconds over all tasks (OPT) broken into large and small instances as well as instances for task T_S and T_L.

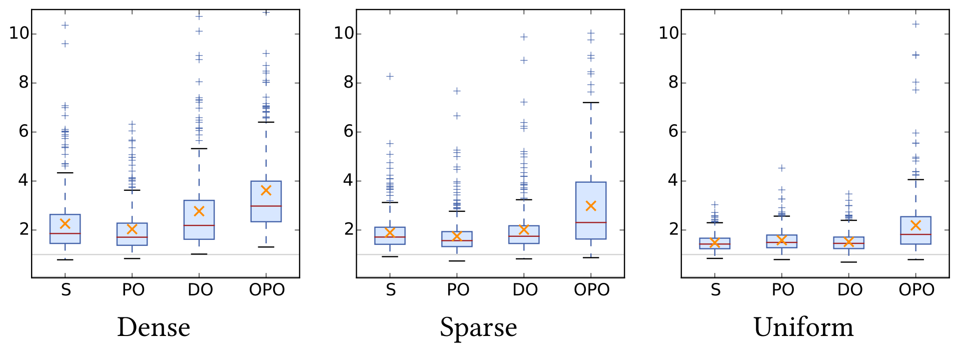


(b) Response times in seconds over all tasks (OPT) broken into dense, sparse and uniform instances as well as all instances.

Figure 11.7: Absolute response times (in seconds) broken down to different parameters. Mean values are indicated by a bold 'x'. The corresponding significances are found in Table 11.5. Smaller values are better than higher values.



(a) Response times in seconds over all correctly processed tasks (CPT) broken into large, small instances as well as instances for task T_S and T_L.



(b) Response times in seconds over all correctly processed tasks (CPT) broken into dense, sparse and uniform instances as well as all instances.

Figure 11.8: Absolute response times (in seconds) broken down to different parameters. Mean values are indicated by a bold 'x'. The corresponding significances are found in Table 11.5. Smaller values are better than higher values.

12 Multi-Sided Boundary Labeling

Abstract. In this chapter, we consider the boundary labeling problem from a theoretical perspective following the model of Bekos et al. [Bek+07]. We assume that we are given a set of n sites inside an axis-parallel rectangle R and a set of n pairwise disjoint rectangular labels that are attached to R from the outside. More precisely, we study the *multi-sided boundary labeling* problem, with labels lying on at least two sides of the enclosing rectangle. The task is to connect the sites to the labels by *po*-leaders. We present a polynomial-time algorithm that computes a crossing-free leader layout if one exists. So far, such an algorithm has only been known for the cases in which labels lie on one side or on two opposite sides of R (here a crossing-free solution always exists). The case where labels may lie on adjacent sides is more difficult.

This chapter is based on and partly taken from joint work with Philipp Kindermann, Ignaz Rutter, Marcus Schaefer, André Schulz and Alexander Wolff[1] [Kin+16, Kin15, Kin+13b, Kin+13a].

12.1 Introduction

Following Bekos et al. [Bek+07], we describe the BoundaryLabeling problem considered in this chapter as follows. We are given an axis-parallel rectangle $R = [0, W] \times [0, H]$, describing the *boundary* of the image, a set $S \subset R$ of n sites $p_1, \ldots, p_n$, within the rectangle R, and a set L of $m \leq n$ axis-parallel rectangles $\ell_1, \ldots, \ell_m$ of equal size, which model the labels. The labels lie in the complement of R and touch the boundary of R. No two labels overlap. We denote an instance of the problem by the triplet (R, S, L). A *solution* of a problem instance is a set of m curves $c_1, \ldots, c_m$ in the interior of R, representing the leaders, that connect sites to labels such that the leaders

a) induce a matching between the labels and (a subset of) the sites,
b) touch the associated labels on the boundary of R.

Following previous work, we do not define labels as the text associated with the sites, but as the empty rectangles into which that text will be placed (during a post-processing step). This approach is justified by our assumption that all label rectangles have the same size.

[1]The chapter is based on a close scientific cooperation with other researchers and contains the parts of [Kin+16] to which the author of this thesis has substantially contributed to. The content of this chapter has also been published in the dissertation by Philipp Kindermann[Kin15].

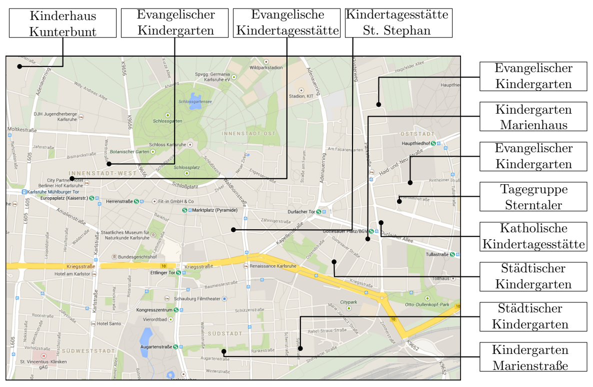

Kinderhaus Kunterbunt
Evangelischer Kindergarten
Evangelische Kindertagesstätte
Kindertagesstätte St. Stephan
Evangelischer Kindergarten
Kindergarten Marienhaus
Evangelischer Kindergarten
Tagegruppe Sterntaler
Katholische Kindertagesstätte
Städtischer Kindergarten
Städtischer Kindergarten
Kindergarten Marienstraße

Figure 12.1: Labeling of kindergartens in Karlsruhe, Germany. The picture shows *po*-leaders with labels on adjacent sides of the map. For better readability, we have simplified the label texts.

A solution is *planar* if the leaders do not intersect. We call an instance *solvable* if a planar solution exists. Note that we do not prescribe which site connects to which label. The endpoint of a curve at a label is called a *port*. We distinguish two versions of the Boundary Labeling problem: either the position of the ports on the boundary of R is fixed and part of the input, or the ports *slide*, i.e., their exact location is not prescribed.

We restrict our solutions to *po*-leaders; see Figure 12.1. Bekos et al. [Bek+10b, Figure 16] observed that not every instance admits a planar solution with *po*-leaders in which all sites are labeled (even if $m = n$).

Previous and Related Work. While in Chapter 10 we have reviewed the preceding research generally, we now take an algorithmic perspective on closely related work. For *po*-labeling, Bekos et al. [Bek+07] gave a simple quadratic-time algorithm for the one-sided case that, in a first pass, produces a labeling of minimum total leader length by matching sites and ports from bottom to top. In a second pass, their algorithm removes all intersections without increasing the total leader length. This result was improved by Benkert et al. [Ben+09] who gave an $O(n \log n)$-time algorithm for the same objective function and an $O(n^3)$-time algorithm for a very general class of objective functions, including, for example, bend minimization. They extend the latter result to the two-sided case (with labels on opposite sides of R), resulting in an $O(n^8)$-time algorithm. For the special case of leader-length minimization, Bekos et al. [Bek+07] gave a simple dynamic program running in $O(n^2)$ time. All these algorithms work both for fixed and sliding ports.

At its core, the boundary labeling problem asks for a non-intersecting perfect (or maximum) matching on a bipartite graph. Note that an instance may have a planar solution, although all of its leader-length minimal matchings have crossings. In fact, the ratio between a length-minimal solution and a length-minimal crossing-free matching can be arbitrarily bad; see Figure 12.2. When connecting points and sites with straight-

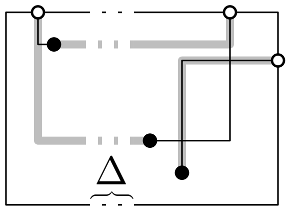


Figure 12.2: Length-minimal solutions may have crossings. By increasing Δ we can make the ratio between the length-minimal matching and the length-minimal crossing-free matching arbitrarily small.

line segments, the minimum Euclidean matching is necessarily crossing-free. For this case an $O(n^{2+\varepsilon})$-time $O(n^{1+\varepsilon})$-space algorithm exists [AES99].

Boundary labeling can also be seen as a graph-drawing problem where the class of graphs to be drawn is restricted to matchings. The restriction concerning the positions of the graph vertices (that is, sites and ports) has been studied for less restricted graph classes under the name *point-set embeddability (PSE)*, usually following the straight-line drawing convention for edges [Gri+91]. For polygonal edges, Bastert and Fekete [BF96] proved that PSE with minimum number of bends or minimum total edge length is NP-hard, even when the graph is a matching. For minimizing the total edge length and the same graph class, Liebling et al. [Lie+95] introduced heuristics and Chan et al. [Cha+13] presented approximation algorithms. Chan et al. also considered paths and general planar graphs. PSE has also been combined with the ortho-geodesic drawing convention [Kat+09], which generalizes *po*-labeling by allowing edges to have more than one bend. The case where the mapping between ports and sites is given has been studied in VLSI layout [RCS86].

For fixed ports, boundary labeling can be modeled as finding an independent set in outerstring graphs. A graph $G = (V, E)$ is an *outerstring graph*, if G forms an intersection graph on curves that lie in a bounded region in such a way that each curve has one end point on the boundary of that region. We obtain G as follows. For each possible site-port pair we create its unique leader λ; the vertices of G are these leaders. Two vertices are adjacent in G if and only if the according leaders intersect. Note that the construction is independent from the shape of the leaders. Obviously, an independent set in G implies a planar boundary labeling. Recently, Keil et al. [Kei+17][2] presented a dynamic programming approach that computes an independent set in $O(k^3)$ time and $O(k^2)$ space, where k denotes the total number of line segments describing the bounding region and the curves. Since we construct $O(n^2)$ leaders, a boundary labeling with fixed ports can be computed in $O(n^6)$ running time and $O(n^4)$ space; the labels may lie on any side of the boundary, and the boundary can be an arbitrary simple polygon.

Contribution and Outline. In the first part of the chapter, we investigate the problem Two-Sided Boundary Labeling with Adjacent Sides where all labels

[2]The prelimary publications [Kin+16, Kin+13b, Kin+13a] of this chapter have been published before.

lie on two *adjacent* sides of R, without loss of generality, on the top and right side. Note that point data often comes in a coordinate system; then it is natural to have labels on adjacent sides (for example, opposite the coordinate axes). We argue that this problem is more difficult than the case where labels lie on opposite sides, which has been studied before: with labels on opposite sides, (a) there is always a solution where all sites are labeled (if $m = n$) and (b) a feasible solution can be obtained by considering two instances of the one-sided case.

We present an algorithm that, given an instance with n labels with fixed ports and n sites, decides whether a planar solution exists where all sites are labeled and, if yes, computes a layout of the leaders; see Section 12.3. Our algorithm uses dynamic programming to "guess" a partition of the sites into the two sets that are connected to the leaders on the top side and on the right side. The algorithm runs in $O(n^2)$ time and uses $O(n)$ space. In [Kin+16] we further present several extensions of our main result[3], which include the label-number maximization problem, sliding ports as well as total leader length minimization.

In the second part of the chapter, we investigate the problems Three-Sided Boundary Labeling and Four-Sided Boundary Labeling where the labels may lie on three or even all four sides of R, respectively. To that end we generalize the concept of partitioning the sites labeled by leaders of different sides. In this way we obtain subinstances that we can solve using the algorithm for the two-sided case. We obtain an algorithm solving the four-sided case in $O(n^9)$ time and $O(n)$ space and an algorithm solving the three-sided case in $O(n^4)$ time and $O(n)$ space.

While for the two-sided and three-sided case our algorithm is clearly faster than the algorithm based on the dynamic programming approach by Keil et al. [Kei+17], for the four sided case our algorithm outperforms that algorithm concerning space consumption.

Notation. We call the labels that lie on the right (left/top/bottom) side of R *right (left/top/bottom) labels*. The *type* of a label refers to the side of R on which it is located. The *type* of a leader (or a site) is simply the type of its label. We assume that no two sites lie on the same horizontal or vertical line, and no site lies on a horizontal or vertical line through a port or an edge of a label.

For a solution $\mathcal{L}$ of a boundary labeling problem, we define several measures that will be used to compare different solutions. We denote the total length of all leaders in $\mathcal{L}$ by $\text{length}(\mathcal{L})$. Moreover, we denote by $|\mathcal{L}|_x$ the total length of all horizontal segments of leaders that connect a left or right label to a site. Similarly, we denote by $|\mathcal{L}|_y$ the total length of the vertical segments of leaders that connect top or bottom labels to sites. Note that generally, $|\mathcal{L}|_x + |\mathcal{L}|_y \neq \text{length}(\mathcal{L})$.

[3]Since the author's contribution to these extensions is not significant, they are not presented in this thesis, but the reader may refer to [Kin+16] for details.

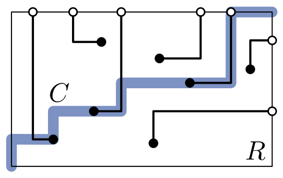


Figure 12.3: An xy-separating curve of a planar solution.

We denote the (uniquely defined) leader connecting a site p to a port t of a label ℓ by $\lambda(p,t)$. We denote the bend of the leader $\lambda(p,t)$ by $\text{bend}(p,t)$. In the case of fixed ports, we identify ports with labels and simply write $\lambda(p,\ell)$ and $\text{bend}(p,\ell)$, respectively.

12.2 Structure of Two-Sided Planar Solutions

In this section, we tackle the two-sided boundary labeling problem with adjacent sides by presenting a series of structural results of increasing strength. We assume that the labels are located on the top and right sides of R. For simplicity, we assume that we have fixed ports. By identifying the ports with their labels, we use L to denote the set of ports of all labels. For sliding ports, we can simply fix all ports to the bottom-left corner of their corresponding labels. First we show that a planar two-sided solution admits a transformation sustaining planarity such that the result of the transformation can be split into two one-sided solutions by constructing an xy-monotone, rectilinear curve from the top-right to the bottom-left corner of R; see Figure 12.3. Afterwards, we provide a necessary and sufficient criterion to decide whether there exists a planar solution for a given separation. This will form the basis of our dynamic programming algorithm, which we present in Section 12.3.

Lemma 12.1. *Consider a solution $\mathcal{L}$ for (R,S,L) and let $S' \subseteq S$ be sites of the same type. Let $L' \subseteq L$ be the set of labels of the sites in S'. Let $K \subseteq R$ be a rectangle that contains all bends of the leaders of S'. If the leaders of $S \setminus S'$ do not intersect K, then we can rematch S' and L' such that the resulting solution $\mathcal{L}'$ has the following properties: (i) all intersections in K are removed, (ii) there are no new intersections of leaders outside of K, (iii) $|\mathcal{L}'|_x = |\mathcal{L}|_x$, $|\mathcal{L}'|_y = |\mathcal{L}|_y$, and (iv) $\text{length}(\mathcal{L}') \leq \text{length}(\mathcal{L})$.*

Proof. Without loss of generality, we assume that S' contains top sites; the other cases are symmetric. We first prove that, no matter how we change the assignment between S' and L', new intersection points can arise only in K. This enables us to construct the required solution.

Claim 12.1. *Let $\ell, \ell' \in L'$ and $p, p' \in S$ such that ℓ labels p and ℓ' labels p'. Changing the matching by rerouting p to ℓ' and p' to ℓ does not introduce new intersections outside of K.*

Let $K' \subseteq K$ be the rectangle spanned by $\text{bend}(p,\ell)$ and $\text{bend}(p',\ell')$. When rerouting, we replace $\lambda(p,\ell) \cup \lambda(p',\ell')$ restricted to the boundary of K' by its complement with

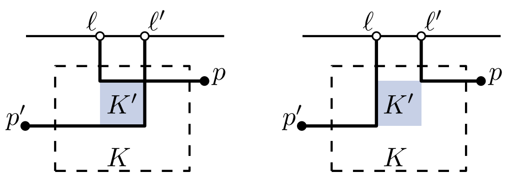


Figure 12.4: Illustration of the proof of Lemma 12.1. Rerouting $\lambda(p, \ell)$ and $\lambda(p', \ell')$ to $\lambda(p, \ell')$ and $\lambda(p', \ell)$ changes leaders only on the boundary of K'

respect to the boundary of K'; see Figure 12.4 for an example. Thus, any changes concerning the leaders occur only in K'. The statement of the claim follows.

Since any rerouting can be seen as a sequence of pairwise reroutings, the above claim shows that we can rematch L' and S' arbitrarily without running the risk of creating new conflicts outside of K. To resolve the conflicts inside K, we use the length-minimization algorithm for one-sided boundary labeling by Benkert et al. [Ben+09], with the sites and ports outside K projected onto the boundary of K. Thus, we obtain a solution $\mathcal{L}'$ satisfying properties (i)–(iv). □

Definition 12.1. *We call an xy-monotone, rectilinear curve connecting the top-right to the bottom-left corner of R an xy-*separating curve*. We say that a planar solution to* Two-Sided Boundary Labeling with Adjacent Sides *is xy-*separated *if and only if there exists an xy-separating curve C such that*
a) the sites that are connected to the top side and all their leaders lie on or above C
b) the sites that are connected to the right side and all their leaders lie below C.

It is not hard to see that a planar solution is not xy-separated if there exists a site p that is labeled to the right side and a site q that is labeled to the top side with $x(p) < x(q)$ and $y(p) > y(q)$. There are exactly four patterns in a possible planar solution that satisfy this condition; see Figure 12.5. In Lemma 12.2, we show that these patterns are the only ones that can violate xy-separability.

Lemma 12.2. *A planar solution is xy-separated if and only if it does not contain any of the patterns P1–P4 in Figure 12.5.*

Proof. Obviously, the planar solution is not xy-separated if one of these patterns occurs. Let us assume that none of these patterns exists. We construct an xy-monotone curve C from the top-right corner of R to its bottom-left corner. We move to the left whenever possible, and down only when we reach the x-coordinate of a site p that is connected to the top, or when we reach the x-coordinate of a port of a top label, labeling a site p. If we have to move down, we move down as far as necessary to avoid the corresponding leader, namely down to the y-coordinate of p. Finally, when we reach the left boundary of R, we move down to the bottom-left corner of R. If C is free of crossings, then we have found an xy-separating curve. (For an example, see curve C in Figure 12.3.)

Assume for a contradiction, that a crossing arises during the construction, and consider the topmost such crossing. Note that, by the construction of C, crossings

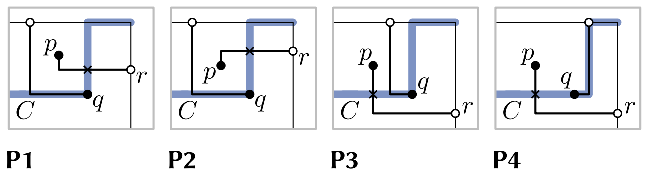


Figure 12.5: A planar solution that contains any of the above four patterns P1–P4 is not xy-separated.

can only occur with leaders that connect a site p to a right port r. We distinguish two cases, based on whether the crossing occurs on a horizontal or a vertical segment of the curve C.

If C is crossed on a vertical segment, then this segment belongs to a leader connecting a site q to a top port t, and we have reached the x-coordinate of either the port or the site. Had we, however, reached the x-coordinate of the port, this would imply a crossing between $\lambda(p, r)$ and $\lambda(q, t)$. Thus, we have reached the x-coordinate of q. This means that p lies to the left of and above q, and we have found one of the patterns P1 and P2; see Figure 12.5.

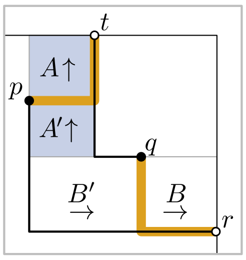
If C is crossed on a horizontal segment, then p must lie above r. Otherwise, there would be another crossing of C with the same leader, which is above the current one. This would contradict the choice of the topmost crossing. Consider the previous segment of C, which is responsible for reaching the y-coordinate of the current segment. This vertical segment belongs to a leader connecting a site q to a top port t. Since leaders do not cross, q is to the right of p, and the crossing on C implies that q is below p. We have found one of the patterns P3 and P4; see Figure 12.5. □

Observe that patterns P1 and P2 can be transformed into patterns P3 and P4, respectively, by mirroring the instance diagonally. Next, we prove constructively that, by rerouting pairs of leaders, any planar solution can be transformed into an xy-separated planar solution.

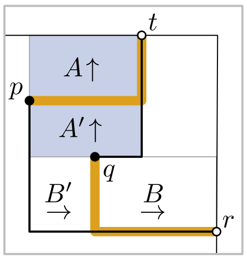
Proposition 12.1. *If there exists a planar solution $\mathcal{L}$ to* Two-Sided Boundary Labeling with Adjacent Sides*, then there exists an xy-separated planar solution $\mathcal{L}'$ with* $\text{length}(\mathcal{L}') \leq \text{length}(\mathcal{L})$*, $|\mathcal{L}'|_x \leq |\mathcal{L}|_x$, and $|\mathcal{L}'|_y \leq |\mathcal{L}|_y$.*

Proof. Let $\mathcal{L}$ be a planar solution of minimum total leader length. We show that $\mathcal{L}$ is xy-separated. Assume, for the sake of contradiction, that $\mathcal{L}$ is not xy-separated. Then, by Lemma 12.2, $\mathcal{L}$ contains one of the patterns P1–P4. Without loss of generality, we can assume that the pattern is of type P3 or P4. Otherwise, we mirror the instance diagonally.

Consider all patterns (p, q) in $\mathcal{L}$ of type P3 or P4 such that p is a right site (with port r) and q is a top site (with port t). Among all such patterns, consider the ones where p is rightmost and among these pick one where q is bottommost. Let A be the rectangle spanned by p and t; see Figure 12.6. Let A' be the rectangle spanned



(a) Pattern P3.



(b) Pattern P4.

Figure 12.6: Types (top = $\uparrow$ / right = $\longrightarrow$) of the sites inside rectangles A, A', B, and B'.

by bend(q, t) and p. Let B be the rectangle spanned by q and r. Let B' be the rectangle spanned by q and bend(p, r). Then we claim the following:

(i) Sites in the interiors of A and A' are connected to the top.
(ii) Sites in the interiors of B and B' are connected to the right.

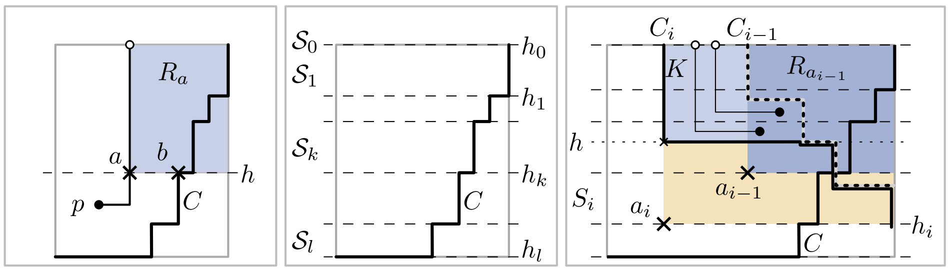
Property (i) is due to the choice of p as the rightmost site involved in such a pattern. Similarly, property (ii) is due to the choice of q as the bottommost site that forms a pattern with p. This settles our claim.

Our goal is to change the labeling by rerouting p to t and q to r, which decreases the total leader length, but may introduce crossings. We then use Lemma 12.1 to remove the crossings without increasing the total leader length. Let $\mathcal{L}''$ be the labeling obtained from $\mathcal{L}$ by rerouting p to t and q to r. We have $|\mathcal{L}''|_y \leq |\mathcal{L}|_y - (y(p) - y(q))$ and $|\mathcal{L}''|_x = |\mathcal{L}|_x - (x(q) - x(p))$. Moreover, length$(\mathcal{L}'') \leq$ length$(\mathcal{L}) - 2(y(p) - y(q))$, as at least twice the vertical distance between p and q is saved; see Figure 12.6. Since the original labeling was planar, crossings may only arise on the horizontal segment of $\lambda(p, t)$ and on the vertical segment of $\lambda(q, r)$.

By properties (i) and (ii), all leaders that cross the new leader $\lambda(p, t)$ have their bends inside A', and all leaders that cross the new leader $\lambda(q, r)$ have their bends inside B'. Thus, we can apply Lemma 12.1 to the rectangles A' and B' to resolve all new crossings. The resulting solution $\mathcal{L}'$ is planar and has length less than length$(\mathcal{L})$. This is a contradiction to the choice of $\mathcal{L}$. □

Since every solvable instance of Two-Sided Boundary Labeling with Adjacent Sides admits an xy-separated planar solution, it suffices to search for such a solution. Moreover, an xy-separated planar solution that minimizes the total leader length is a solution of minimum length. In Lemma 12.3 we provide a necessary and sufficient criterion to decide whether, for a given xy-monotone curve C, there is a planar solution that is separated by C. We denote the region of R above C by R_T and the region of R below C by R_R. We do not include C in either R_T or R_R, so these regions are open at C.

For any point $a \in R$, we define the rectangle R_a, spanned by the top-right corner of R and a. We define R_a such that it is closed but does *not* contain its top-left corner.



(a) **(b)** **(c)**

Figure 12.7: The strip condition. (a) The horizontal strip condition of b is satisfied by a. (b) The horizontal segments of C partition the strips $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_k$. (c) Constructing a planar labeling from a sequence of valid rectangles.

In particular, we consider the port of a top label as contained in R_a, only if it is not the upper left corner.

A rectangle R_a is *valid* if the number of sites of S above C that belong to R_a is at least as large as the number of ports on the top side of R_a. The central idea is that the labels on the top side of a valid rectangle R_a can be connected to the sites in R_a by leaders that are completely contained inside that rectangle. We are now ready to present the *strip condition.*

Condition 12.1. *The* horizontal strip condition *of the point $b \in C$ is satisfied if there exists a point $a \in R_\mathrm{T}$ with $y(a) = y(b)$ and $x(a) \leq x(b)$ such that R_a is valid.*

Without loss of generality we may assume that the curve C is rectilinear. The condition is named after the horizontal segments through points in C.

We now prove that, for a given xy-monotone curve C connecting the top-right corner to the bottom-left corner of R, there exists a planar solution in R_T for the top labels if and only if all points of C satisfy the strip condition.

Lemma 12.3. *Let C be an xy-monotone curve from the top-right corner of R to the bottom-left corner of R. Let $S' \subseteq S$ be the sites that are in R_T. There is a planar solution that uses all top labels of R to label sites in S' in such a way that all leaders are in R_T if and only if each point of C satisfies the strip condition.*

Proof. For the proof we call a region $S \subseteq R$ *balanced* if it contains the same number of sites as it contains ports. To show that the conditions are necessary, let $\mathcal{L}$ be a planar solution for which all top leaders are above C. Consider a point $b \in C$. If $y(p) \geq y(b)$ for all sites $p \in S$, rectangle R_a with $a = (0, y(b))$ is clearly valid, and thus the strip condition for b is satisfied. Hence, assume that there is a site $p \in S$ with $y(p) < y(b)$ that is labeled by a top label; see Figure 12.7(a). Then, the vertical segment of this leader crosses the horizontal line h through b. Let a denote the rightmost such crossing of a leader of a site in S' with h. We claim that R_a is valid. To see this, observe that all sites of S' top-right of a are contained in R_a. Since no leader may cross the vertical segment defining a, the number of sites in $R_a \cap R_\mathrm{T}$ is balanced, i.e., R_a is valid.

Conversely, we show that if the conditions are satisfied, then a corresponding planar solution exists. For each horizontal segment of C consider the horizontal line through the segment. We denote the part of these lines within R by $h_1, \dots, h_l$, respectively, and let h_0 be the top side of R. The line segments $h_1, \dots, h_l$ partition R_T into l strips, which we denote by $\mathcal{S}_1, \dots, \mathcal{S}_l$ from top to bottom, such that strip $\mathcal{S}_i$ is bounded by h_i from below for $i = 1, \dots, l$; see Figure 12.7(b). Additionally, we define $\mathcal{S}_0$ to be the empty strip that coincides with h_0. Let $\mathcal{S}_k$ be the last strip that contains sites of S'. For $i = 0, \dots, k-1$, let a'_i denote the rightmost point of $h_i \cap R_\mathrm{T}$ such that $R_{a'_i}$ is valid. Such a point exists since the leftmost point of $h_i \cap C$ satisfies the strip condition. We define a_i to be the point on $h_i \cap R_\mathrm{T}$, whose x-coordinate is $\min_{j \le i}\{x(a'_j)\}$. Note that R_{a_i} is a valid rectangle, as, by definition, it completely contains some valid rectangle $R_{a'_j}$ with $x(a'_j) = x(a_i)$. Also by definition the sequence formed by the points a_i has decreasing x-coordinates, i.e., the R_{a_i} grow to the left; see Figure 12.7(c).

We prove inductively that, for each $i = 0, \dots, k$, there is a planar labeling $\mathcal{L}_i$ that matches the labels on the top side of R_{a_i} to points contained in R_{a_i}, in such a way that there exists an xy-monotone curve C_i from the top-left to the bottom-right corner of R_{a_i} that separates the labeled sites from the unlabeled sites without intersecting any leaders. Then $\mathcal{L}_k$ is the required labeling.

For $i = 0$, $\mathcal{L}_0 = \emptyset$ is a planar solution. Consider a strip $\mathcal{S}_i$ with $0 < i \le k$; see Figure 12.7(c). By the induction hypothesis, we have a curve C_{i-1} and a planar labeling $\mathcal{L}_{i-1}$, which matches the labels on the top side of $R_{a_{i-1}}$ to the sites in $R_{a_{i-1}}$ above C_{i-1}. To extend it to a planar solution $\mathcal{L}_i$, we additionally need to match the remaining labels on the top side of R_{a_i} and construct a corresponding curve C_i. Let S_i denote the set of unlabeled sites in R_{a_i}. By the validity of R_{a_i}, this number is at least as large as the number of unused ports at the top side of R_{a_i}. We arbitrarily match these ports to the topmost sites of S_i that are not labeled in $\mathcal{L}_{i-1}$. We denote the resulting labeling by $\mathcal{L}'_i$. We observe that no leader of $\mathcal{L}'_i$ crosses the curve C_{i-1}, and hence such leaders cannot cross leaders in $\mathcal{L}_{i-1}$. Let h be the topmost horizontal line such that all labeled sites of $\mathcal{L}'_i$ lie above h. Further, let K be the rectangle that is spanned by the top-left corner of $R_{a_{i-1}}$ and the intersection of h with the left side of R_{a_i}. Since the ports of $\mathcal{L}'_i$ lie on the top side of K, any leader's bend of $\mathcal{L}'_i$ lies in K. We apply Lemma 12.1 on $\mathcal{L}'_i$ to obtain a planar labeling $\mathcal{L}''_i$, which has no crossings with $\mathcal{L}_{i-1}$. Hence, the set $\mathcal{L}_i = \mathcal{L}''_i \cup \mathcal{L}_{i-1}$ is the required labeling.

It remains to construct the curve C_i. For this, we start at the top-left corner of R_{a_i} and move down vertically, until we have passed all labeled sites. We then move right until we either hit C_{i-1} or the right side of R. In the former case, we follow C_{i-1} until we arrive at the right side of R. Finally, we move down until we arrive at the bottom-right corner of R_{a_i}. Note that all labeled sites are above C_i, unlabeled sites are below C_i, and no leader is crossed by C_i. This is true since we first move below the new leaders and then follow the previous curve C_{i-1}. □

A symmetric strip condition (with vertical strips) can be obtained for the right region R_R of a partitioned instance. The characterization is completely symmetric.

In the following, we observe two properties of the strip condition. The first observation states that the horizontal strip condition at (x, y) is independent of the exact shape of the curve between the top-right corner r of R and (x, y), as long as the number of sites above the curve remains the same. This is crucial for using dynamic programming to test the existence of a suitable curve. The second observation states that the horizontal strip condition can only be violated when the curve passes the x-coordinate of a top site. This enables us to discretize the problem.

Observation 12.1. *The horizontal strip condition for a point $a \in C$ depends only on the number of sites in R_a above C, in the following sense: Let C and C' be two xy-monotone curves from r to a with u sites in R_a above C and C', respectively. Then, a satisfies the strip condition for C if and only if it satisfies the strip condition for C'.*

Observation 12.2. *Let $a, b \in C, x(a) \leq x(b)$ such that there is no top site ℓ with $x(a) < x(\ell) \leq x(b)$. Then, a satisfies the horizontal strip condition for C if and only if b satisfies the horizontal strip condition for C.*

Symmetric statements hold for the vertical strip condition. In the following, we say that a point (x, y) on a curve C satisfies the *strip condition* if it satisfies both the horizontal and the vertical strip condition.

12.3 Algorithm for the Two-Sided Case

How can we find an xy-monotone curve C that satisfies the strip conditions? For that purpose we only consider xy-monotone curves contained in some graph G that is dual to the rectangular grid induced by the sites and ports of the given instance. Note that this is not a restriction since all leaders are contained in the grid induced by the sites and ports. Thus, every xy-monotone curve that does not intersect the leaders can be transformed into an equivalent xy-monotone curve that lies on G.

When traversing an edge e of G, we pass the x- or y-coordinate of exactly one entity of our instance; either a site (*site event*) or a port (*port event*). When passing a site, the position of the site relative to e (above/below e or right/left of e) decides whether the site is connected to the top or to the right side. Clearly, there is an exponential number of possible xy-monotone traversals through the grid. In the following, we describe a dynamic program that finds an xy-separating curve in $O(n^3)$ time.

Let m_R and m_T be the numbers of ports on the right and top side of R, respectively. Also, let $N = n + m_\mathrm{T} + 2$ and $M = n + m_\mathrm{R} + 2$, then the grid G has size $N \times M$. We define the grid points as $G(s,t)$, $0 \le s \le N$, $0 \le t \le M$ with $G(0,0)$ being the bottom-left and $r := G(N,M)$ being the top-right corner of R. Finally, let $G_x(s) := x(G(s,0))$ and $G_y(t) := y(G(0,t))$.

For each grid point (s,t) that is neither on the topmost row nor on the rightmost column, we define four boxes $B^{\uparrow}(s,t), B^{\downarrow}(s,t), B^{\leftarrow}(s,t)$ and $B^{\rightarrow}(s,t)$ as follows; see Figure 12.8 for an illustration.

1. $B^{\uparrow}(s,t) = \{(x,y) \in R \mid G_x(s) \le x \le G_x(s+1) \wedge y \ge G_y(t)\}$
2. $B^{\downarrow}(s,t) = \{(x,y) \in R \mid G_x(s) \le x \le G_x(s+1) \wedge y \le G_y(t)\}$
3. $B^{\leftarrow}(s,t) = \{(x,y) \in R \mid G_y(t) \le y \le G_y(t+1) \wedge x \le G_x(s)\}$
4. $B^{\rightarrow}(s,t) = \{(x,y) \in R \mid G_y(t) \le y \le G_y(t+1) \wedge x \ge G_x(s)\}$

We define a table $T[(s,t),u,b]$ that assigns to each grid position (s,t) and number of points u and b a Boolean value. We define $T[(s,t),u,b]$ to be `true` if and only if there exists an xy-monotone curve C satisfying the following conditions.

(i) Curve C starts at r and ends at $G(s,t)$.
(ii) Inside the rectangle spanned by r and $G(s,t)$, there are u sites of S above C and b sites of S below C.
(iii) For each grid point on C, the strip condition holds.

These conditions together with Proposition 12.1 and Lemma 12.3 imply that the instance admits a planar solution if and only if $T[(0,0),u,b] = \texttt{true}$ for some u and b.

We define a Boolean function $X[(s,t),u,b]$ that is true if and only if the strip condition at (s,t) is satisfied for some xy-monotone curve C (and thus by Observation 12.1 for all such curves) from r to $G(s,t)$ with u sites above and b sites below C. The following lemma gives a recurrence for T, which is essentially a disjunction of two values, each of which is determined by distinguishing three cases.

Lemma 12.4. *For $s = N$ and $t = M$, it holds that $T[(s,t),0,0] = \mathit{true}$. For $s \in [0, N-1]$ and $t \in [0, M-1]$, it holds that*

$$T[(s,t),u,b] = \begin{array}{c} \left\{ \begin{array}{lcl} T[(s+1,t),u,b] \wedge X[(s,t),u,b] & & L \cap B^{\uparrow}(s,t) \neq \emptyset \\ T[(s+1,t),u-1,b] & \mathit{if} & S \cap B^{\uparrow}(s,t) \neq \emptyset \\ T[(s+1,t),u,b-1] & & S \cap B^{\downarrow}(s,t) \neq \emptyset \end{array} \right\} \\ \vee \\ \left\{ \begin{array}{lcl} T[(s,t+1),u,b] \wedge X[(s,t),u,b] & & L \cap B^{\rightarrow}(s,t) \neq \emptyset \\ T[(s,t+1),u,b-1] & \mathit{if} & S \cap B^{\rightarrow}(s,t) \neq \emptyset \\ T[(s,t+1),u-1,b] & & S \cap B^{\leftarrow}(s,t) \neq \emptyset \end{array} \right\}. \end{array}$$

Proof. We show equivalence of the two terms. Let C be an xy-monotone curve from r to (s,t). Let e be the last segment of C and let $C' = C - e$. Since C is xy-monotone, C' ends either at the grid point $(s+1,t)$ or at $(s,t+1)$. Without loss of generality, we

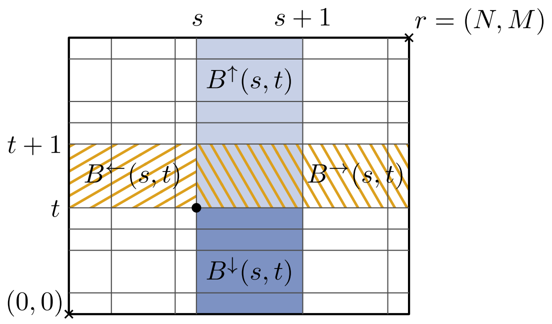


Figure 12.8: The four boxes $B^{\uparrow}(s,t)$, $B^{\downarrow}(s,t)$, $B^{\leftarrow}(s,t)$ and $B^{\rightarrow}(s,t)$ defined by grid point (s,t).

assume that C' ends at $(s+1,t)$. We show that $T[(s,t),u,b] = \texttt{true}$ if and only if the first term of the right hand side is `true`. Analogous arguments apply for C' ending at $(s, t+1)$ and the second term. Note that, by construction, property (i) is satisfied for C and C'.

We distinguish cases based on whether the traversal along the segment e from $(s+1,t)$ to (s,t) is a port event or a site event.

Case 1: Traversal of e is a port event. Since e passes a port, all sites that lie in the rectangle spanned by r and $G(s,t)$ also lie in the rectangle spanned by r and $G(s+1,t)$. Thus, the numbers u and b of such sites above and below C is the same as the numbers of sites above and below C', respectively. Hence, property (ii) holds for C if and only if it holds for C'.

Because C' is a subset of C, the strip condition holds for every point of C if and only if it holds for every point of C' and for (s,t). Thus, property (iii) is satisfied for C if and only if it is satisfied for C' and $X[(s,t),u,b] = \texttt{true}$.

Case 2: Traversal of e passes a site p. For property (iii), observe that, since the traversal of e is a site event, the strip conditions for (s,t) and $(s+1,t)$ are equivalent by Observation 12.2.

For property (ii), note that, except for p, the sites that lie in the rectangle spanned by r and $G(s,t)$ also lie in the rectangle spanned by r and $G(s+1,t)$. If p lies above e, there are u sites above and b sites below C if and only if there are $u-1$ sites above and b sites below C', respectively. Symmetrically, if p lies below e, there are u sites above and b sites below C if and only if there are u sites above and $b-1$ sites below C', respectively. In either case, C satisfies condition (ii) if and only if C' does. □

Clearly, the recurrence from Lemma 12.4 can be used to compute T in polynomial time via dynamic programming. Note that it suffices to store u, as the number of sites below the curve C can directly be derived from u and all sites that are contained in the rectangle spanned by r and $G(s,t)$. Thus, in the following we work with $T[(s,t),u]$. The running time crucially relies on the number of strip conditions that need to be checked. We show that after a $O(n^2)$ preprocessing phase, such queries can be answered in $O(1)$ time.

To implement the test of the strip conditions, we use a table B_{T}, which stores in $B_{\mathrm{T}}[s, t]$ how large a deficit of sites to the right can be compensated by sites above and to the left of $G(s, t)$. That is, $B_{\mathrm{T}}[s, t]$ is the maximum value k such that there exists a rectangle $K_{B_{\mathrm{T}}[s,t]}$ with lower right corner $G(s, t)$ whose top side is bounded by the top side of R, and that contains k more sites in its interior, than it has ports on its top side. Once we have computed this matrix, it is possible to query the strip condition in the dynamic program that computes T in $O(1)$ time as follows: Assume we have an entry $T[(s, t), u]$, and we wish to check its strip condition. Consider a curve C from r to $G(s, t)$ such that u sites are above C. The strip condition is satisfied if and only if $u + B_{\mathrm{T}}[s, t]$ is at least as large as the number of top ports to the right of $G(s, t)$. This is true if the rectangle spanned by the lower left corner of $K_{B_{\mathrm{T}}[s,t]}$ and r contains at least $u + B_{\mathrm{T}}[s, t]$ sites, which is an upper bound on the number of ports on the top side of that rectangle.

We now show how to compute B_{T} in $O(n^2)$ time. We compute each row separately, starting from the left side. We initialize $B_{\mathrm{T}}[0, t] = 0$ for $t = 0, \dots, M$, since in the final column, no deficit can be compensated. The matrix B can be filled by a horizontal sweep. The entry $B_{\mathrm{T}}[s, t]$ can be derived from the already computed entry $B_{\mathrm{T}}[s - 1, t]$. If the step from $s - 1$ to s is a site event, the amount of the deficit we can compensate increases by 1. If it is a port event the amount of the deficit we can compensate decreases by 1. Moreover, the compensation potential never goes below 0. We obtain

$$B_{\mathrm{T}}[s,t] = \begin{cases} B_{\mathrm{T}}[s-1,t] + 1 & \text{if step is site event,} \\ \max\{B_{\mathrm{T}}[s-1,t] - 1, 0\} & \text{if step is port event.} \end{cases}$$

The table can be clearly filled out in $O(n^2)$ time. A similar matrix B_{R} can be computed for the vertical strips. Altogether, this yields an algorithm for Two-Sided Boundary Labeling with Adjacent Sides that runs in $O(n^3)$ time and uses $O(n^3)$ space. However, the entries of each row and column of T depend only on the previous row and column, which allows us to reduce the storage requirement to $O(n^2)$. Using Hirschberg's algorithm [Hir75], we can still backtrack the dynamic program and find a solution corresponding to an entry in the last cell in the same running time. We have the following theorem.

Theorem 12.1. *Two-Sided Boundary Labeling with Adjacent Sides can be solved in $O(n^3)$ time using $O(n^2)$ space.*

Our next goal is to improve the performance of our algorithm by reducing the number of dimensions of the table T by 1. As a first step, we show that for any search position $\mathbf{c} = (s, t)$, the set of all u with $T[\mathbf{c}, u] = \mathtt{true}$ is an interval.

Lemma 12.5. *Let $T[\mathbf{c}, u] = T[\mathbf{c}, u'] = \mathtt{true}$ with $u < u'$. Then $T[\mathbf{c}, u''] = \mathtt{true}$ for $u \leq u'' \leq u'$.*

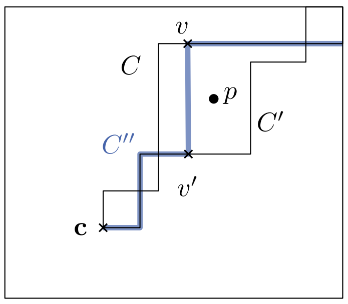


Figure 12.9: Proof sketch of Lemma 12.5.

Proof. Let C be a curve corresponding to the entry $T[\mathbf{c}, u]$. That is C connects r to $\mathbf{c}$ such that any point on C satisfies the strip condition. Similarly, let C' be a curve corresponding to $T[\mathbf{c}, u']$; see Figure 12.9.

Since u and u' differ, there is a rightmost site p, such that p is below C and above C'. Let v and v' be the grid points of C and C' that are immediately to the left of p. Note that v is above v' since C is above p and C' is below it. Consider the curve C'' that starts at r and follows C until v, then moves down vertically to v', and from there follows C' to p. Obviously C'' is an xy-monotone curve, and it has above it the same sites as C', except for p, which is below it. Thus there are $u'' = u' - 1$ sites above C'' in the rectangle spanned by p and r. If all points of C'' satisfy the strip condition, then this implies $T[\mathbf{c}, u''] = \texttt{true}$.

We show that indeed the strip condition is satisfied for any point on C''. Let C_1 be the subcurve of C'' that connects r to v, let C_2 be the segment vv' and let C_3 be the subcurve of C'' that connects v' to $\mathbf{c}$. Since C_1 is also a subcurve of C and it starts at r, it directly follows that any point of C_1 satisfies the strip condition. For the points on C_2 we can argue as follows. Since C_2 lies below C and any point of C satisfies the horizontal strip condition, any point of C_2 must satisfy the horizontal strip condition. Analogously, because C_2 lies above C' and any point of C' satisfies the vertical strip condition, each point of C_2 must satisfy the vertical strip condition. Finally, since C_3 is a subcurve of C', any point of C' satisfies the strip condition and any point of C_1 and C_2 satisfies the strip condition, it directly follows that any point of C_3 satisfies the strip condition. □

Using Lemma 12.5, we can reduce the dimension of the table T by 1. It suffices to store at each entry $T[\mathbf{c}]$ the boundaries of the u-interval. This reduces the amount of storage to $O(n^2)$ without increasing the running time. Using Hirschberg's algorithm, the storage for T even decreases to $O(n)$. Tables B_T and B_R still have size $O(n^2)$, however.

Our next goal is to reduce the running time to $O(n^2)$. An entry in $B_\mathrm{T}[s, t]$ tells us which deficits can be compensated. This can also be interpreted as a lower bound on the number of sites a curve from r to $G(s, t)$ must have above it, in order to satisfy

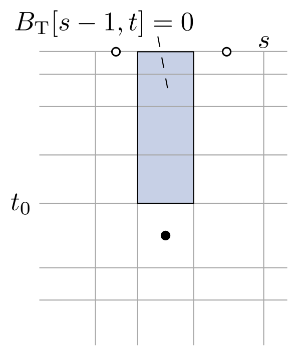


Figure 12.10: The gap t_0 is defined such that we have $B_\mathrm{T}[s-1,t] = 0$ for any $t \geq t_0$, and $B_\mathrm{T}[s-1,t] > 0$ for any $t < t_0$.

the horizontal strip condition. Namely, let $\tau_{s,t}$ denote the number of ports on the top side of the rectangle spanned by $G(s,t)$ and r. Then $u \geq \tau_{s,t} - B_\mathrm{T}[s,t]$ is equivalent to satisfying the horizontal strip condition for the strip directly above $G(s,t)$. Similarly, the corresponding entry $B_\mathrm{R}[s,t]$ gives a lower bound on the number of sites below such a curve, which in turn, together with the number of sites contained in the rectangle spanned by $G(s,t)$ and r implies an *upper bound* on the number of sites above the curve. Thus, B_T, B_R, and the information on how many sites, top ports and right ports are in the rectangle spanned by $G(s,t)$ and r together imply a lower and an upper bound, and thus an interval of u-values, for which the horizontal and vertical strip conditions at $G(s,t)$ is satisfied. Hence the program can simply intersect this interval with the union of the intervals obtained from $T[(s,t) - \mathbf{\Delta c}]$, where $\mathbf{\Delta c}$ has exactly one non-zero entry, which is 1. Consequently, the amount of work per entry of T is still $O(1)$. Note that by Lemma 12.5 the result of this computation is again an interval.

Now we turn to the space consumption. Hirschberg's algorithm [Hir75] immediately reduces the space consumption of T to $O(n)$. We would like to apply the same trick to B_T and to B_R. Recall that B_T is computed from left to right and B_R from bottom to top. Unfortunately, this is opposite to the order we use for computing T, where we proceed from top-right to bottom-left. We can fix this problem by running the dynamic programs for computing B_T and B_R backwards, by precomputing the entries of B_T and B_R on the top and right side, and then running the updates backwards. This allows us to use Hirschberg's algorithm, and the algorithms can run in a synchronized manner such that at any point in time the required data is available, using only $O(n)$ space.

A new issue, however, appears. The update $B_\mathrm{T}[s,t] = \max\{B_\mathrm{T}[s-1,t] - 1, 0\}$ is not easily reversible. When running the dynamic program backwards, it is not clear whether $B_\mathrm{T}[s,t] = 0$ implies $B_\mathrm{T}[s-1,t] = 0$ or $B_\mathrm{T}[s-1,t] = 1$ at a port step. To remedy this issue, fix a column s of the table corresponding to a port event and consider the circumstances under which $B_\mathrm{T}[s-1,t] - 1 = -1$, i.e., $B_\mathrm{T}[s-1,t] = 0$. This implies that, for any rectangle K with lower right corner $G(s-1,t)$ whose top side is contained in the top side of R, there are at most as many sites in K as there are ports in the top side of K. Assume that this is the case for some fixed value t_0, i.e., $B_T[s-1,t_0]$. Since the possible rectangles for an entry $B_T[s-1,t]$ with $t \geq t_0$ contain at most as many sites as the ones for $B_T[s-1,t_0]$, this implies $B_T[s-1,t_0] = B_T[s-1,t] = 0$ for all $t \geq t_0$. If on the other hand, t_0 is such that $B_T[s-1,t_0] > 0$, then the rectangles corresponding to $B_T[s-1,t]$ for $t < t_0$ contain at least as many sites as the ones for $B_T[s-1,t_0]$, and we have $B_T[s-1,t] \geq B_T[s-1,t_0]$ for $t < t_0$. Thus, there is a single gap t_0 such that, for any $t \geq t_0$, we have $B_T[s-1,t] = 0$ and, for any $t < t_0$, we have $B_T[s-1,t] > 0$; see Figure 12.10. Storing this gap for each column s that is a port event allows us to efficiently reverse the dynamic program. Note that storing one value per column only incurs $O(n)$ space overhead. Of course, the same approach works for the dynamic program computing B_R. Overall, we have shown the following theorem.

Theorem 12.2. *Two-Sided Boundary Labeling with Adjacent Sides can be solved in $O(n^2)$ time using $O(n)$ space.*

12.4 The Three- and Four-Sided Cases

In this section, we also allow labels on the bottom and the left side of R. In order to solve an instance of the three- and four-sided case, we adapt the techniques we developed for the two-sided case. We assume that the ports are fixed and the number of labels and sites is equal. In Section 12.4.1 we first analyze the structure of planar solutions obtaining a result similar to Proposition 12.1. In Sections 12.4.2 and 12.4.3, we present algorithms for the three- and four-sided cases.

12.4.1 Structure of Three- and Four-Sided Planar Solutions

Similar to our approach to two-sided boundary labeling, we pursue the idea that if there exists a planar solution, then we can also find a planar solution such that there are four xy-monotone curves connecting the four corners of R to a common point o, and such that these curves separate the leaders of the different label types from each other; see Figure 12.11. To that end, we first show that leaders of left and right labels can be separated vertically and leaders of top and bottom labels can be separated horizontally. Afterwards, we apply the result of Lemma 12.2 in order to resolve the remaining overlaps, e.g., between top and right leaders. We first introduce some notions.

Definition 12.2. *A planar solution for the four-sided boundary labeling problem is*

(i) x-separated *if there exists a vertical line ℓ such that the sites that are labeled to the left side are to the left of ℓ and the sites that are labeled to the right side are to the right of ℓ, and*

(ii) y-separated *if there exists a horizontal line ℓ such that the sites that are labeled to the top side are above ℓ and the sites that are labeled to the bottom side are below ℓ.*

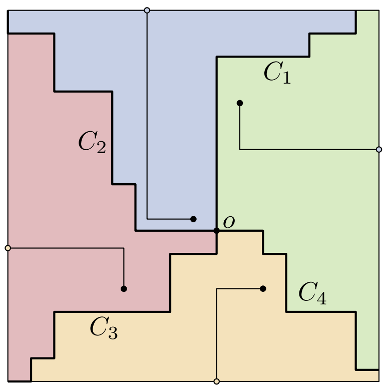


Figure 12.11: The curves C_1, C_2, C_3 and C_4 meeting at the point o partition the rectangle into four regions.

A left leader λ and a right leader λ' *overlap* if $x(\text{bend}(\lambda)) > x(\text{bend}(\lambda'))$. Analogously, a bottom leader λ and a top leader λ' *overlap* if $y(\text{bend}(\lambda)) > y(\text{bend}(\lambda'))$. Hence, a planar solution $\mathcal{L}$ is both x-separated and y-separated if and only if no left and right leaders overlap, and no bottom and top leaders overlap. We are now ready to prove that we can always find a planar solution that is both x-separated and y-separated, if a solution exists.

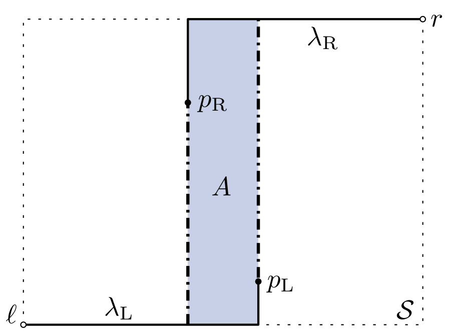
Lemma 12.6. *If there exists a planar solution for the four-sided boundary labeling problem, then there exists a planar solution $\mathcal{L}$ that is both x-separated and y-separated.*

Proof. Among all planar solutions let $\mathcal{L}$ be one that minimizes $|\mathcal{L}|_x + |\mathcal{L}|_y$. We prove that then $\mathcal{L}$ is x- and y-separated by showing that otherwise we could reroute some leaders and obtain a planar solution $\mathcal{L}'$ with $|\mathcal{L}'|_x + |\mathcal{L}'|_y < |\mathcal{L}|_x + |\mathcal{L}|_y$.

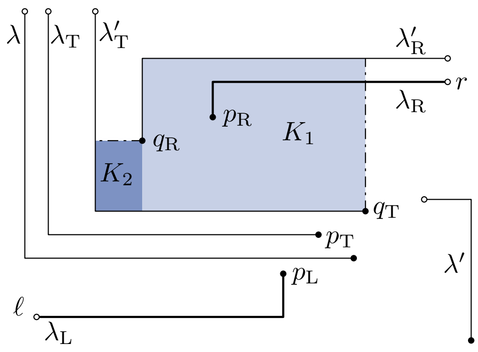
Assume that $\mathcal{L}$ is not x-separated. Symmetric arguments hold for the case that $\mathcal{L}$ is not y-separated. Then there exist sites p_R and p_L with $x(p_R) < x(p_L)$, such that p_R is labeled by a right port r, and p_L is labeled by a left port ℓ; see Figure 12.12(a). Without loss of generality, assume that the horizontal segment of $\lambda_R = \lambda(p_R, r)$ is above the horizontal segment of $\lambda_L = \lambda(p_L, \ell)$, otherwise we mirror the instance vertically.

We choose p_L and p_R as a closest pair in the sense that the horizontal segments of their leaders have minimum vertical distance among all such pairs. Let A be the rectangle spanned by $\text{bend}(\lambda_L)$ and $\text{bend}(\lambda_R)$. By the minimality of p_L and p_R, that rectangle can only be intersected by top and bottom leader, but not by left or right leaders. If no such leader intersects A, we reroute p_R to the port of λ_L and p_L to the port of λ_R, which decreases $|\mathcal{L}|_x$ without increasing $|\mathcal{L}|_y$; see Figure 12.12(a). It does not introduce any crossings.

In the following, we assume that some leaders intersect A. Without loss of generality we assume that there is a top leader λ_T that intersects A; otherwise we rotate the instance by 180°. We denote its site by p_T. Let $\mathcal{S}$ be the rectangle spanned by the ports ℓ and r; see Figure 12.12(a). Depending on the leaders intersecting $\mathcal{S}$, we distinguish two cases. Note that in particular λ_T intersects $\mathcal{S}$.



(a) A is empty.



(b) Case 1.

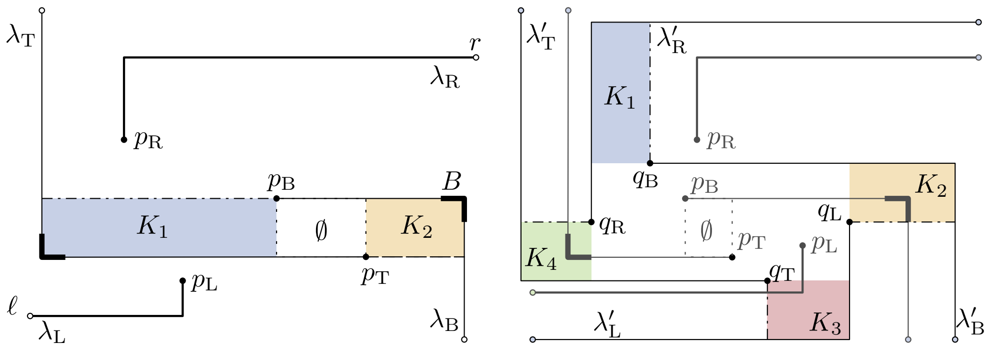
Figure 12.12: Different constellations of leaders intersecting the rectangle A. (a) The rectangle A is empty. (b) The rectangle A is intersected by a top leader. A is not explicitly illustrated, but spanned by bend(λ_{R}) and bend(λ_{L}).

Case 1: For any top leader λ intersecting $\mathcal{S}$ and for any bottom leader λ' intersecting $\mathcal{S}$ such that λ and λ' overlap, the site of λ lies to the left of the site of λ'; see Figure 12.12(b). Let q_{R} denote the bottommost site that is connected by a right leader, and that lies in the rectangle spanned by bend(λ_{T}) and p_{R}. Since p_{R} lies in that rectangle, the site q_{R} exists. We denote the leader of q_{R} by λ'_{R}. Further, let q_{T} be the topmost site that is connected by a top leader and that lies in the rectangle spanned by bend(λ'_{R}) and the bottom-right corner of R. Since p_{T} lies in that rectangle, the site q_{T} exists. We denote its leader by λ'_{T}.

We now define two rectangles that we use to reroute leaders such that $|\mathcal{L}|_x + |\mathcal{L}|_y$ is decreased and arising crossings can be resolved. The rectangle K_1 is spanned by bend(λ'_{R}) and q_{T}, and the rectangle K_2 is spanned by bend(λ'_{T}) and q_{R}.

Claim 12.2.
(1) K_1 is only intersected by right leaders whose bends are contained in K_1,
(2) K_2 is only intersected by top leaders whose bends are contained in K_2, and
(3) K_1 and K_2 are internally disjoint.

Assuming that the claim holds, we can reroute the sites as follows; we illustrate this rerouting by dash-dotted lines in Figure 12.12(b). The site q_{T} is rerouted to the port of λ'_{R} creating crossings only on the right side of K_1. The site q_{R} is rerouted to the port of λ'_{T} creating crossings only on the top side of K_2. Each rerouting decreases either $|\mathcal{L}|_x$ or $|\mathcal{L}_y|$ increasing the other one. Further, only crossings between leaders of the same type are created. We apply Lemma 12.1 to resolve the conflicts without increasing $|\mathcal{L}|_x$ or $|\mathcal{L}_y|$. In the remainder of this case we show that the stated claim holds.



(a) Case 2.1 **(b)** Case 2.2

Figure 12.13: Different constellations of top leaders intersecting the rectangle A. A is not explicitly illustrated, but spanned by bend(λ_R) and bend(λ_L).

First, we show that K_1 is only intersected by right leaders whose bends lie in K_1. It is not intersected by any bottom leader, because such a leader would overlap λ'_T, and its site would lie to the left of q_T—a contradiction to the assumption of this case. It is not intersected by any left leader, because such a leader would intersect λ'_T. It is not intersected by any top leader, because such a leader would either intersect λ'_R or contradict the choice of λ'_T. Hence, K_1 can only be intersected by right leaders. Further, all those leaders have their bend in K_1, because the bottom-right corner is a site connected by a top leader. That leader would be intersected if a right leader intersecting K_1 had its bend outside of K_1.

Next, we show that K_2 is only intersected by top leaders whose bends lie in K_2. It is not intersected by any right leader, because such a leader would contradict the choice of λ'_R or intersect λ_T. It is not intersected by any bottom leader, because such a leader would overlap λ'_T, and its site would lie to the left of q_T—a contradiction to the assumption of this case. It is not intersected by any left leader, because such a leader would intersect λ'_T. Hence, K_2 can only be intersected by top leaders. Further, all those leaders have their bend in K_2, because the top-right corner is a site connected by a right leader.

Finally, the rectangles K_1 and K_2 are internally disjoint, because K_1 lies to the right of the vertical line through q_R, while K_2 lies to the left of that line.

Case 2: There exist a top leader λ_T intersecting $\mathcal{S}$ and a bottom leader λ_B intersecting $\mathcal{S}$ such that they overlap and the site of λ_T lies to the right of the site of λ_B; see Figure 12.13(a). Among all such pairs we choose λ_T and λ_B such that their horizontal segments have minimal vertical distance. We denote the site of λ_T by p_T and the site of λ_B by p_B. Due to the choice of λ_T and λ_B, the open rectangle that is spanned by p_B and p_T is intersected by no leader. The open rectangle spanned by bend(λ_T)

and bend(λ_B) is denoted by B. Depending on the sites that are contained in B, we distinguish four cases.

Case 2.1: The rectangle B contains no sites that are connected by left of right leaders; see Figure 12.13(a). Let K_1 be the rectangle spanned by bend(λ_T) and p_B, and let K_2 be the rectangle spanned by bend(λ_B) and p_T. While K_1 is only intersected by top leaders, K_2 is only intersected by bottom leaders. Further, both rectangles are disjoint. We reroute p_B to the port of λ_T and p_T to the port of λ_B. Obviously, this decreases $|\mathcal{L}|_y$ without increasing $|\mathcal{L}|_x$. By applying Lemma 12.1, we resolve the arising conflicts.

Case 2.2: The rectangle B contains sites that are connected by left leaders as well as sites that are connected by right leaders; see Figure 12.13(b). Let q_R be the bottommost site in B that is connected to the right. We denote the leader of q_R by λ'_R. Let q_B be the leftmost site with $y(q_\mathrm{B}) \geq y(p_\mathrm{B})$ and $x(q_\mathrm{B}) \leq x(p_\mathrm{B})$ that is connected to the bottom. Since p_B also satisfies these requirements, the site q_B exists. We denote the leader of q_B by λ'_B. Let q_L be the topmost site in B that is connected to the left. We denote the leader of q_L by λ'_L. Finally, let q_T be the rightmost site with $y(q_\mathrm{T}) \leq y(p_\mathrm{T})$ and $x(q_\mathrm{T}) \geq x(p_\mathrm{T})$ that is connected to the top. Since p_T also satisfies these requirements, the site q_T exists. We denote the leader of q_T by λ'_T.

We now define four rectangles that we use to reroute leaders such that $|\mathcal{L}|_x + |\mathcal{L}|_y$ is decreased and arising crossings can be resolved. The rectangle K_1 is spanned by bend(λ'_R) and q_B, the rectangle K_2 is spanned by bend(λ'_B) and q_L, the rectangle K_3 is spanned by bend(λ'_L) and q_T, and the rectangle K_4 is spanned by bend(λ'_T) and q_R. Note that the rectangles K_3 and K_4 are rotationally symmetric to K_1 and K_2, respectively.

Claim 12.3.
(1) K_1 is only intersected by right leaders whose bends are contained in K_1,
(2) K_2 is only intersected by bottom leaders whose bends are contained in K_2,
(3) K_3 is only intersected by left leaders whose bends are contained in K_3,
(4) K_4 is only intersected by top leaders whose bends are contained in K_4, and
(5) K_1, K_2, K_3 and K_4 are pairwise internally disjoint.

Assuming that the claim holds, we can reroute the sites in a circular fashion as follows; we illustrate the rerouting as dash-dotted lines in Figure 12.13(b). The site q_B is rerouted to the port of λ'_R creating crossings only on the right side of K_1. The site q_L is rerouted to the port of λ'_B creating crossings only on the bottom side of K_2. The site q_T is rerouted to the port of λ'_L creating crossing only on the left side of K_3. Finally, the site q_R is rerouted to the port of λ'_T creating crossings only on the top side of K_4. Each rerouting decreases either $|\mathcal{L}|_x$ or $|\mathcal{L}_y|$ without increasing the other one. Further, only crossings between leaders of the same type are created. We apply Lemma 12.1 to resolve the conflicts. In the remainder of this case we show that the stated claim holds.

First, we show that K_1 is only intersected by right leaders whose bends lie in K_1. This rectangle is not intersected by any bottom leader, because q_B is the leftmost site with $y(q_B) \geq y(p_B)$ and $x(q_B) \leq x(p_B)$ that is connected to the bottom. It is not intersected by any top leader, because such a leader would intersect λ'_R whose site lies below q_B. Finally, it is not intersected by any left leader, because such a leader would intersect λ'_T whose site lies to the right of q_B. Hence, only right leaders intersect K_1. In particular, all those leaders have their bend in K_1, because the bottom-right corner of K_1 is the site of a bottom leader. That leader would be intersected if a right leader intersecting K_1 had its bend outside of K_1. Since K_3 is rotationally symmetric to K_1, we can use symmetric arguments to prove that K_3 is only intersected by left leaders whose bends are contained in K_3

Next, we show that K_2 is only intersected by bottom leaders whose bends lie in K_2. This rectangle is not intersected by any left leader, because such a leader would contradict the choice of q_L. It is also not intersected by any top leader, because such a leader would intersect λ'_R or contradict the choice of λ_T and λ_B. Finally, it cannot be intersected by any right leader, because such a leader would intersect λ'_B. Hence, K_2 is only intersected by bottom leaders. Further, all those leaders have their bend in K_2, because the bottom-left corner of K_2 is a site connected to a left leader. That leader would be intersected if a bottom leader intersecting K_2 had its bend outside of K_2. Since K_4 is rotationally symmetric to K_2, we can use symmetric arguments to prove that K_4 is only intersected by top leaders whose bends are contained in K_4.

Finally, we show that the rectangles K_1, K_2, K_3 and K_4 are pairwise internally disjoint. For a site p let $v(p)$ denote the vertical line through p and let $h(p)$ denote the horizontal line through p. By construction we have that $h(q_B)$ lies above $h(q_T)$, K_1 lies above $h(q_B)$, and K_3 lies below $h(q_T)$. Hence, the rectangles K_1 and K_3 are internally disjoint. Analogously, we have that $v(q_L)$ lies to the right of $v(q_R)$, K_2 lies to the right of $v(q_L)$, and K_4 lies to the left of $v(q_R)$. Hence, the rectangles K_2 and K_4 are internally disjoint. Further, the sites q_L and q_R lie in between $h(q_B)$ and $h(q_T)$, because both lie in B. Consequently, K_1 and K_3 do not intersect K_2 and K_4, respectively.

Case 2.3: The rectangle B contains only sites connected by right leaders. We apply the same procedure as in the previous case. However, we do not need to consider left leaders. Hence, K_3 is removed and K_2 is the rectangle that is spanned by bend(λ'_B) and p_T. By the choice of B, the rectangle K_2 is only intersected by right leaders whose bend is contained in K_2. Further, the remaining rectangles K_1, K_2 and K_4 are pairwise internally disjoint. The reroutings are again done in a circular fashion decreasing $|\mathcal{L}|_x + |\mathcal{L}|_y$. Finally, we apply Lemma 12.1 to resolve crossings.

Case 2.4: The rectangle B contains only sites connected by left leaders. This case can be handled analogously to the previous case by mirroring the instance vertically. ◻

This lemma shows that, when searching for a planar solution of the labeling problem, we can restrict ourselves to solutions that are x-separated and y-separated. Let $\mathcal{L}$ denote such a solution, and let ℓ_v and ℓ_h be the lines separating the sites labeled by left and right labels, and the ones labeled by top and bottom labels, respectively.

Let $o \in R$ denote the intersection of ℓ_v and ℓ_h, called *center point*. Let $r_1, \ldots, r_4$ denote the corners of R, named in counterclockwise ordering, and such that r_1 is the top-right corner. Consider the rectangles that are spanned by o and r_i for $i = 1, \ldots, 4$. Each of them contains only two types of leaders. For example, the top-right rectangle contains only top and right leaders. An x- and y-separated planar solution is *partitioned* if, for each rectangle spanned by o and one of the corners r_i of R, there exists an xy-monotone curve C_i from r_i to o that separates the two different types of leaders contained in that rectangle; see Figure 12.11. Our next step is to show that a planar solution can be transformed into a partitioned solution without increasing $|\mathcal{L}|_x$ and $|\mathcal{L}|_y$.

Proposition 12.2. *If there exists a planar solution $\mathcal{L}$ for* Four-Sided Boundary Labeling*, then there exists a partitioned solution $\mathcal{L}'$.*

Proof. By Lemma 12.6, we can assume that $\mathcal{L}$ is x- and y-separated. Let o be the center point as defined above and let ℓ_v be the vertical line through o. We show how to ensure that the area K of R right of ℓ_v admits an xy-monotone curve from the top-right corner of R to o that separates the top leaders from the right leaders inside K. The remaining cases are symmetric.

Essentially, we proceed as in the proof of Proposition 12.1 to remove obstructions of types (P1)–(P4); see Figure 12.5. We note that in the rerouting, we only shorten vertical segments of top leaders and right segments of right leaders; hence the solution remains x- and y-separated. Moreover, in each step we decrease both $|\mathcal{L}|_x$ and $|\mathcal{L}|_y$. Hence, after finitely many steps all patterns between top and right leaders have been removed without creating new patterns with other types of leaders.

After all patterns have been removed, an xy-monotone curve connecting the top-right corner of R to o, separating the top labels from the right labels, can be found as in the proof of Lemma 12.2. □

12.4.2 Algorithm for the Three-Sided Case

In the three-sided case, we assume that the ports of the given instance I are located on three sides of R; without loss of generality, on the left, top and right side of R. Basically, we solve a three-sided instance by splitting the instance into two two-sided L-shaped instances that can be solved independently; see Figure 12.14(a).

Let G be the dual of the grid that is induced by the sites and ports of the given instance. The idea is that each grid point s of G induces two two-sided L-shaped instances with some useful properties. We will show that there is a planar solution for I if and only if there is a grid point s of G such that its induced two-sided instances both have planar solutions. Thus, considering all $O(n^2)$ grid points of G the problem reduces to solve those L-shaped instances of the two-sided case. By means of a simple adaption of the dynamic program presented in Section 12.3 we solve these instances in $O(n^2)$ time achieving $O(n^4)$ running time in total.

In the following, we call horizontal and vertical lines through grid points of G *horizontal* and *vertical grid lines*, respectively. We now define the two two-sided L-shaped instances I_s and I'_s of a grid point s of G formally. To that end, let R_1 be the rectangle that is spanned by the top-right corner of R and s, and let $R_2(p)$ be the rectangle that is spanned by a point p on the horizontal grid line h through s and the bottom-right corner of R; see Figure 12.14(a). The instance $I_s(p)$ contains all sites and ports in $R_1 \cup R_2(p)$ and $I'_s(p)$ contains all sites and ports in $R \setminus (R_1 \cup R_2(p))$. We say that $I_s(p)$ and $I'_s(p)$ are *balanced* if all right ports lie in $R_1 \cup R_2(p)$, all left ports lie in $R \setminus (R_1 \cup R_2(p))$ and $R_1 \cup R_2(p)$ contains the same number of sites as it contains ports. Since the number of ports and sites in I is equal, this directly implies that $R \setminus (R_1 \cup R_2(p))$ contains the same number of sites as it contains ports. In particular, the choice of balanced instances $I_s(p)$ and $I'_s(p)$ for a grid point s of G is unique with respect to the contained sites and ports; only the location of p might differ. We can therefore write I_s and I'_s for balanced instances and R_1 and R_2 for their defining rectangles. For any solution of I_s and any solution of I'_s, we require that all leaders are completely contained in $R_1 \cup R_2$ and in $R \setminus (R_1 \cup R_2)$, respectively. The next lemma states that a three-sided instance I has a planar solution if and only if it can be partitioned into two two-sided L-shaped instances that have planar solutions. To that end let h_s denote the horizontal grid line through s. Figure 12.14 illustrates the lemma.

Lemma 12.7. *There is a planar solution $\mathcal{L}$ for a three-sided instance I if and only if there is a grid point s of G with balanced instances I_s and I'_s over rectangles (R_1, R_2), an xy-monotone curve C from the top-right corner to the bottom-left corner of R and an xy-monotone curve C' from the top-left corner to the bottom-right corner of R such that*

1. *each point on C satisfies the strip condition with respect to the ports and sites in I_s,*
2. *C contains the top-left corner of R_2 and the intersection of h_s with the left segment of R,*
3. *each point on C' satisfies the strip condition with respect to the ports and sites in I'_s,*
4. *C' contains the top-left corner of R_2 and the intersection of h_s with the right segment of R.*

Proof. First, assume that s, I_s, I'_s, (R_1, R_2), C and C' exist as required; see Figure 12.14(b). The curve C partitions $R_1 \cup R_2$ into two regions; we denote the region above C by A_1 and the region below C by A_2. By Lemma 12.3, there is a planar solution $\mathcal{L}_1$ for the sites and ports in A_1 such that all leaders of $\mathcal{L}_1$ lie in A_1. Since C contains the top-left corner of R_2 and does not cross h_s until it reaches the intersection point of h_s with the left segment of R, we know that all leaders of $\mathcal{L}_1$ are contained in $R_1 \cup R_2$. Analogously, there is a planar solution $\mathcal{L}_2$ for the sites and ports in A_2 such that all leaders of $\mathcal{L}_2$ lie in A_2. Consequently, we can combine $\mathcal{L}_1$ and $\mathcal{L}_2$ into a planar solution $\mathcal{L}_s$ for the sites and ports in I_s. Using symmetric arguments, we obtain a planar solution $\mathcal{L}'_s$ for I'_s. As I_s and I'_s are defined over complementary areas, the solutions $\mathcal{L}_s$ and $\mathcal{L}'_s$ can be combined into a planar solution of I.

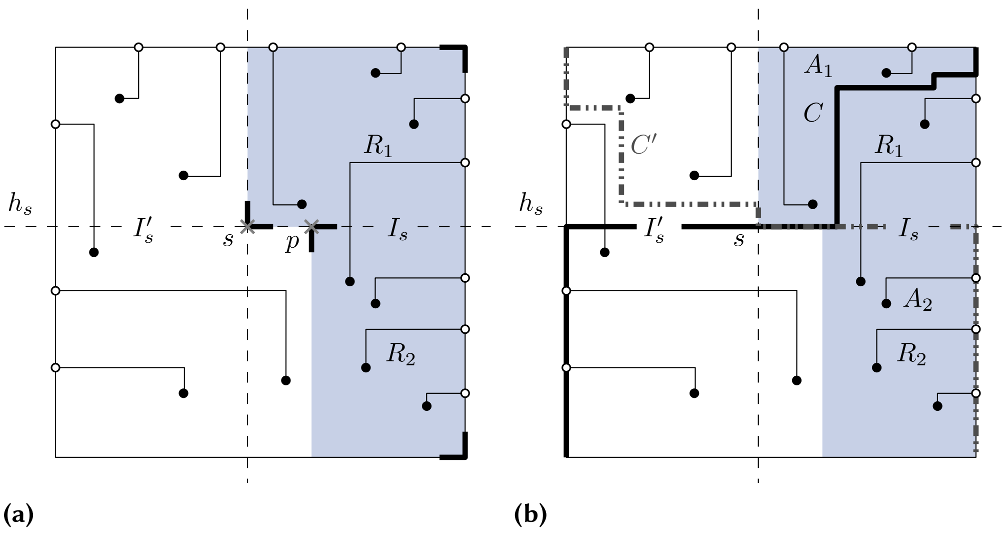


Figure 12.14: (a) The three-sided instance partitioned into two two-sided L-shaped instances I_s and I'_s. The instances are induced by the grid point s of G and are balanced. (b) Illustration of the proof for Lemma 12.7. Assuming that the grid point s of G, the balanced instances I_s and I'_s, and the curves C and C' are given, a planar solution for the whole instance can be constructed.

Assume that there is a planar solution $\mathcal{L}$ for a three-sided instance I; see Figure 12.15. First, note that we can imagine an instance of Three-Sided Boundary Labeling as a degenerated instance of Four-Sided Boundary Labeling with no bottom ports. Thus, Proposition 12.2 also holds for the three-sided case, when assuming that the four xy-monotone curves partitioning the solution meet on the bottom segment of R. Hence, without loss of generality, we assume that $\mathcal{L}$ is also partitioned by four xy-monotone curves C_1, C_2, C_3 and C_4. In particular, let C_1 denote the curve that starts at the top-right corner of R and let C_2 denote the curve that starts at the top-left corner of R; see Figure 12.15(a). The curves C_3 and C_4 are completely contained in the bottom side of R and can therefore be omitted. We first show how to construct the grid point s and the instances I_s and I'_s such that they are balanced. Afterwards, we explain how to obtain C and C' from C_1 and C_2, respectively. Finally, we prove that each point on C and C' satisfies the strip condition with respect to I_s and I'_s, respectively.

Let λ_T be the top leader in $\mathcal{L}$ with the longest vertical segment of all top leaders in $\mathcal{L}$. In case the site of λ_T lies to the right of bend(λ_T), let v be the rightmost vertical grid line that lies to the left of λ_T, and otherwise if the site of λ_T lies to the left of bend(λ_T), let v be the leftmost vertical grid line that lies to the right of λ_T. Furthermore, let h be the topmost horizontal grid line that lies below bend(λ_T); see Figure 12.15(a). Due to the choice of h and v all top leaders lie above h and none of them intersects h or v.

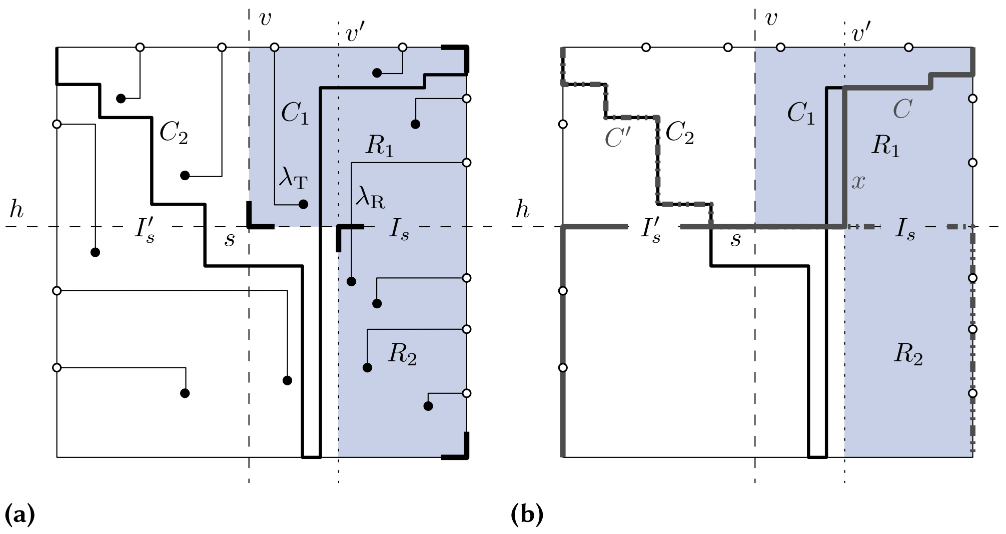


Figure 12.15: Illustration of the proof for Lemma 12.7. It is assumed that the partitioned planar solution $\mathcal{L}$ for the three-sided instance is given. (a) By Proposition 12.2 we can assume that $\mathcal{L}$ is partitioned by the curves C_1 and C_2. The extremal top leader λ_T induces the site s and the extremal right leader λ_R induces the line v'. (b) Based on C_1, C_2, h and v', the curves C and C' can be constructed such that they do not cross any leader of $\mathcal{L}$.

Furthermore, no right or left leader of $\mathcal{L}$ intersects v above h. The desired grid point s is then the intersection point of h and v.

Now, let λ_R be the right leader in $\mathcal{L}$ with longest horizontal segment among all right leaders in $\mathcal{L}$ and let v' be the rightmost vertical grid line that lies to the left of bend(λ_R). Note that v' cannot be intersected by a left or a right leader, because both leader types are x-separated. We define R_1 to be the rectangle that is spanned by the top-right corner of R and s. Also, we define R_2 to be the rectangle spanned by the bottom-right corner of R and the intersection point of v' and h. The instance I_s is defined by $R_1 \cup R_2$ and the instance I'_s by $R \setminus (R_1 \cup R_2)$. We show that I_s and I'_s are balanced. To that end, we prove that a leader of $\mathcal{L}$ is either completely contained in $R_1 \cup R_2$ or in $R \setminus (R_1 \cup R_2)$, that $R_1 \cup R_2$ contains only right and top leaders, and that $R \setminus (R_1 \cup R_2)$ contains only left and top leaders.

Due to the choice of v', all right leaders lie to the right of v'. Moreover, all right leaders whose site or port lies above h, must lie to the right of v, because by definition of v no right leader intersects v above h (otherwise it would intersect λ_T), and because otherwise C_1 could not be an xy-monotone curve separating right and top leaders. Thus, all right leaders lie in $R_1 \cup R_2$. For left leaders we can argue similarly. Since left and right leaders of $\mathcal{L}$ are x-separated, all left leaders lie to the left of v'. All left leaders whose site or port lies above h, must lie to the left of v, because by definition of v no

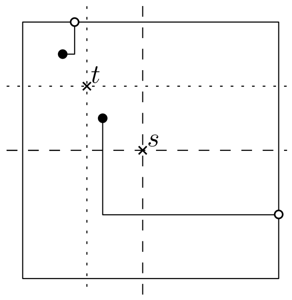


Figure 12.16: There are no balanced instances I_s and I'_s for the grid point s. However, by Lemma 12.7 there must be another grid point t with balanced instances I_t and I'_t if the instance has a planar solution.

left leader intersects v above h, and because otherwise C_2 could not be an xy-monotone curve separating left from top leaders. Thus, all left leaders lie in $R \setminus (R_1 \cup R_2)$. Finally, consider the top leaders in $\mathcal{L}$. By definition of h and v, none of the top leaders intersects h or v. In particular all top leaders lie above h and cannot intersect R_2. Consequently, each top leader is either contained in R_1 or in $R \setminus (R_1 \cup R_2)$. This concludes the argument that I_s and I'_s are balanced.

We are left with the construction of the curves C and C'; see Figure 12.15(b). The curve C is derived from C_1 as follows. Starting at the top-right corner of R, the curve C coincides with C_1 until C_1 intersects h or v' above h. If C intersects v' above h, it follows v' downwards until it hits h. Then, in both cases, it follows h until h intersects the left segment of R. Finally, C follows the left segment of R to the bottom-left corner of R. The curve C' is constructed symmetrically.

By construction, C contains the top-left corner of R_2 and the intersection point of h with the left segment of R. Symmetrically, C' contains the top-left corner of R_2 and the intersection point of h with the right segment of R. We finally show that each point on C satisfies the strip condition with respect to the sites and ports in I_s. Using symmetric arguments we can prove the analogous statement for C' and I'_s.

By the previous reasoning, we know that each leader of $\mathcal{L}$ either lies completely inside or completely outside of $R_1 \cup R_2$. Each leader that lies in $R_1 \cup R_2$ is either a top or a right leader and does not intersect C. Otherwise, if such a leader intersected C, it would also intersect C_1 or the segment x of v' that is contained in C. In particular, x cannot be intersected by any leader because it lies to the left of all right leaders and below C_1. Thus, the leaders in $R_1 \cup R_2$ form a planar solution for I_s without intersecting C. Hence, the claim directly follows from Lemma 12.3. □

Our approach now works as follows. For each grid point s of G we compute the instances I_s and I'_s such that they are balanced. Then, by Lemma 12.7, we can apply our algorithm presented in Section 12.3 in order to solve I_s and I'_s independently. To that end, we slightly adapt the dynamic program such that it only considers curves satisfying the properties required by Lemma 12.7. If both instances can be solved, we combine these solutions into one solution and return that solution as the final result.

Otherwise, we continue with the next grid point of G. If all grid points of G have been explored without finding a planar solution, the algorithm decides that there is no planar solution.

Note that it may happen that, for a grid point s, there are no balanced instances I_s and I'_s; for an example see Figure 12.16. However, in that case, if I has a solution, we also know by Lemma 12.7 that there is another grid point t such that for t we find balanced instances. Hence, we can refrain from considering s.

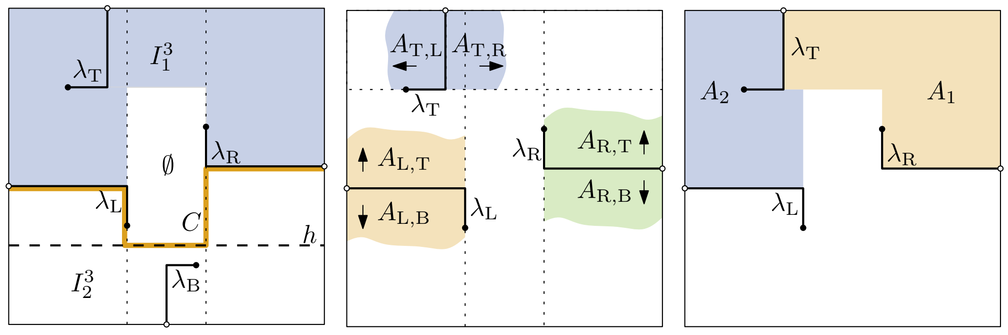
Creating the two instances I_s and I'_s for a grid point s takes linear time, if we assume that the sites are sorted by their x-coordinates. By Theorem 12.2 we then need $O(n^2)$ time and $O(n)$ space to solve I_s and I'_s. Consequently, we need $O(n^2)$ time and $O(n)$ space to process a single grid point s. Since we consider $O(n^2)$ grid points, the following theorem follows.

Theorem 12.3. *Three-Sided Boundary Labeling can be solved in $O(n^4)$ time using $O(n)$ space.*

12.4.3 Algorithm for the Four-Sided Case

In this section, we consider the case that the ports lie on all four sides of R. The main idea is to seek a partitioned solution, which exists by Proposition 12.2. For a given partitioned solution $\mathcal{L}$, we call a leader *extremal* if all other leaders of the same type in $\mathcal{L}$ have shorter orthogonal segments; recall that the *orthogonal* segment of a *po*-leader is the segment connecting the bend to the port. The algorithm consists of two steps. First, we explore all choices of (non-overlapping) extremal leaders λ_L and λ_R for the left and the right side of R, respectively, plus a horizontal line h that separates the top leaders and the bottom leaders. This information splits the instance into two independent three-sided instances; see Figure 12.17(a). There are, however, two crucial differences from a usual three-sided instance. First, one side of the instance is not a straight-line segment but an x-monotone orthogonal curve C that is defined by λ_L, λ_R and h. Second, the extremal positions of λ_L and λ_R imply a separation of the points that are labeled from the left and the right side. Let I_1^3 be the three-sided instance above C and let I_2^3 be the three-sided instance below C. The algorithm solves I_1^3 and I_2^3 independently from each other. If for at least one of the two instances there is no solution, the algorithm continues with the next choice of λ_L, λ_R and h. Otherwise, it combines the planar solutions of I_1^3 and I_2^3 into one planar solution and returns this solution. In case that all choices of λ_L, λ_R and h have been explored without finding a solution, the algorithm returns that there is no planar solution.

We next describe how to solve the three-sided instance I_1^3. A symmetric approach can be applied to I_2^3. The algorithm explores all choices of the extremal leader λ_T for the top side of R. This extremal leader partitions the instance into two two-sided subinstances I_1^2 and I_2^2 as follows. Let $A_{\mathrm{T,R}}$ be the rectangle that is spanned by bend(λ_T)



(a) **(b)** **(c)**

Figure 12.17: (a) The right leader λ_L, the left leader λ_R and the horizontal line h split the instance into two three-sided instances I_1^3 and I_2^3. (a) Sketch of the areas $A_{T,L}$, $A_{T,R}$, $A_{R,T}$, $A_{R,B}$, $A_{L,T}$ and $A_{L,B}$. (c) The leaders λ_L, λ_R and λ_T split the three-sided instance into two two-sided instances.

and the top-right corner of R; see Figure 12.17(b). Analogously, let $A_{T,L}$ be the rectangle that is spanned by bend(λ_T) and the top-left corner of R. Analogously, for λ_R area $A_{R,T}$ is the rectangle that is spanned by bend(λ_R) and the top-right corner of R, and $A_{R,B}$ to be the rectangle spanned by bend(λ_R) and the bottom-right corner of R. For the leader λ_L we define $A_{L,B}$ to be the rectangle spanned by bend(λ_L) and the bottom-left corner of R, and $A_{L,T}$ to be the rectangle spanned by bend(λ_L) and the top-left corner of R. We assume that the port p of λ_T is only contained in that area $A \in \{A_{T,R}, A_{T,L}\}$ that also contains the site of λ_T. We make analogous assumptions for λ_L and λ_R.

The instance I_1^2 consists of all ports and sites in $A_1 = (A_{R,T} \cup A_{T,R}) \setminus (A_{T,L} \cup A_{R,B})$, and I_2^2 consists of all ports and sites in $A_2 = (A_{L,T} \cup A_{T,L}) \setminus (A_{T,R} \cup A_{L,B})$; see Figure 12.17(c). We solve I_1^2 and I_2^2 independently from each other using the dynamic program introduced in Section 12.3 for each instance. However, we enforce that it only considers xy-monotone curves that exclude top leaders crossing the horizontal line through bend(λ_T), left leaders crossing the vertical line through bend(λ_L) and right leaders crossing the vertical line through bend(λ_R). If for at least one of the two instances there is no solution, the algorithm continues to explore the next choice of λ_T. Otherwise, it combines the solutions of I_1^2 and I_2^2 into one solution and returns the result as the solution of I_1^3. In case that all choices of λ_T have been explored without finding a solution, the algorithm returns that there is no solution for the given three-sided instance. The following lemma shows that the algorithm is correct.

Lemma 12.8. *Given an instance I of* Four-Sided Boundary Labeling, *the following two statements are true.*

1. *If there is no planar solution for I, the algorithm states this.*
2. *If there is a planar solution for I, the algorithm returns such a solution.*

Proof. In case the algorithm returns a solution, it has been constructed from planar solutions of disjoint instances of Two-Sided Boundary Labeling with Adjacent Sides. As the union of these two-sided instances contains all sites and ports of I, the algorithm returns a planar solution of I, which shows the first statement.

Conversely, assume that I has a planar solution $\mathcal{L}$. By Proposition 12.2, we may assume that $\mathcal{L}$ is partitioned. In particular, let λ_T, λ_L, λ_B and λ_R be the extremal leaders in $\mathcal{L}$ of the top, left, bottom and right side of R, respectively, and let h be a horizontal line that separates the top leaders from the bottom leaders.

Obviously, λ_L, λ_R and h split the instance into two three-sided instances I_1^3 and I_2^3. As the algorithm systematically explores all choices of extremal right leaders, extremal left leaders and horizontal lines partitioning the set of sites, it must find λ_L, λ_R and a horizontal line h' that separates the same sets of sites as h. Thus, I_1^3 and I_2^3 are considered by the algorithm.

Let I_1^3 be the instance above the curve defined by λ_L, λ_R and h', and let I_2^3 be the instance below that curve. We now show that the algorithm finds a planar solution for I_1^3. Symmetric arguments hold for I_2^3. As the algorithm explores all choices of extremal top leaders in I_1^3, it also considers λ_T to be the extremal top leader. This leader partitions the area of I_1^3 into the two disjoint areas $A_1 = (A_{\mathrm{R,T}} \cup A_{\mathrm{T,R}}) \setminus (A_{\mathrm{T,L}} \cup A_{\mathrm{R,B}})$ and $A_2 = (A_{\mathrm{L,T}} \cup A_{\mathrm{T,L}}) \setminus (A_{\mathrm{T,R}} \cup A_{\mathrm{L,B}})$; see Figure 12.17(a). It directly follows from the extremal choice of λ_R, λ_T and λ_L that there is no leader in $\mathcal{L}$ that intersects both A_1 to A_2. In particular, no left leader intersects A_1 and no right leader intersects A_2. Thus, A_1 and A_2 split $\mathcal{L}$ into independent planar solutions $\mathcal{L}_1$ and $\mathcal{L}_2$ of two two-sided instances I_1^2 and I_2^2 induced by A_1 and A_2, respectively. Note that the algorithm considers the same two-sided instances independently from each other. As I_1^2 has a solution, namely $\mathcal{L}_1$, we know that the dynamic program finds a solution $\mathcal{L}_1^2$ for I_1^2. In particular, all leaders of $\mathcal{L}_1^2$ lie in A_1.

Applying symmetric arguments for I_2^2, the algorithm yields a planar solution $\mathcal{L}_2^2$ that stays in A_2. Consequently, combining $\mathcal{L}_1^2$ and $\mathcal{L}_2^2$ into one solution yields a planar solution $\mathcal{L}_1^3$ for I_1^3. Analogously, we obtain a planar solution $\mathcal{L}_2^3$ for I_2^3. Obviously, due to the separation by λ_L, λ_R and h', the union of $\mathcal{L}_1^3$ and $\mathcal{L}_2^3$ is also planar, which is the overall solution returned by the algorithm. This proves the second statement of the lemma. □

Let us analyze the running time of the algorithm. Obviously, there are $O(n^5)$ possible combinations of left and right extremal leaders and a horizontal line separating the top and bottom-labeled sites. For each combination, we independently solve two three-sided instances. For such a three-sided instance, we consider $O(n^2)$ choices for the extremal leader λ_T and independently solve two independent two-sided instances with Theorem 12.2 in $O(n^2)$ time. This implies that solving one three-sided instances takes $O(n^4)$ time. Thus, the overall running time is $O(n^9)$. The following theorem summarizes this result.

Theorem 12.4. *Four-Sided Boundary Labeling can be solved in $O(n^9)$ time using $O(n)$ space.*

12.5 Conclusions

In this chapter, we have studied the problem of testing whether an instance of Two-Sided Boundary Labeling with Adjacent Sides admits a planar solution. We have given the first efficient algorithm for this problem, running in $O(n^2)$ time.

Our algorithm can also be used to solve a variety of different extensions of the problem, which we present in [Kin+16]. In that paper we show how to generalize to sliding ports instead of fixed ports without increasing the running time and how to maximize the number of labeled sites such that the solution is planar in $O(n^3 \log n)$ time. We further give an extension to the algorithm that minimizes the total leader length in $O(n^8 \log n)$ time.

With some additional work, our approach can also be used to solve Three-Sided and Four-Sided Boundary Labeling in polynomial time. We have introduced an algorithm solving the three-sided case in $O(n^4)$ time and the four-sided case in $O(n^9)$ time. Also, except for the total leader length minimization, all extensions that are presented in [Kin+16] carry over.

13 An Algorithmic Framework for Label Placement in Figures

Abstract. In this chapter, we introduce a flexible and general approach for external label placement assuming a *contour* of the figure prescribing the possible positions of labels. While most research on external label placement aims for fast labeling procedures for interactive systems, we focus on highest-quality illustrations. Based on interviews with domain experts and a semi-automatic analysis of 202 handmade anatomical drawings, we identify a set of 18 layout quality criteria, naturally not all of equal importance. We design a new geometric label placement algorithm that is based only on the most important criteria. Yet, other criteria can flexibly be included in the algorithm, either as hard constraints not to be violated or as soft constraints whose violation is penalized by a general cost function. We formally prove that our approach yields labelings that satisfy all hard constraints and have minimum overall cost. Introducing several speedup techniques, we further demonstrate how to deploy our approach in practice. In an experimental evaluation on real-world anatomical drawings we show that the resulting labelings are of high quality and can be produced in adequate time.

This chapter is based on and partly taken from joint work with Martin Nöllenburg and Ignaz Rutter. [NNR17].

13.1 Introduction

Atlases of human anatomy play a major role in the education of medical students and the teaching of medical terminology. Such books contain a broad spectrum of filigree and detailed drawings of the human anatomy from different cutaway views. For example, the third volume of the popular human anatomy atlas Sobotta [PW13] contains about 1200 figures on 384 pages. Figure 13.1 is one of these drawings showing a temporal bone (lat. os temporale). The usefulness of the figures essentially relies on the naming of the illustrated components. In order not to spoil the readability of the figure by occluding it with text, external labeling is applied.

In this chapter, we present a flexible and versatile approach for external label placement in figures. We use medical drawings as running example, but occlusion-free label placements are also indispensable for the readability of other highly detailed figures as they occur for example in scientific publications, mechanical engineering and maintenance manuals.

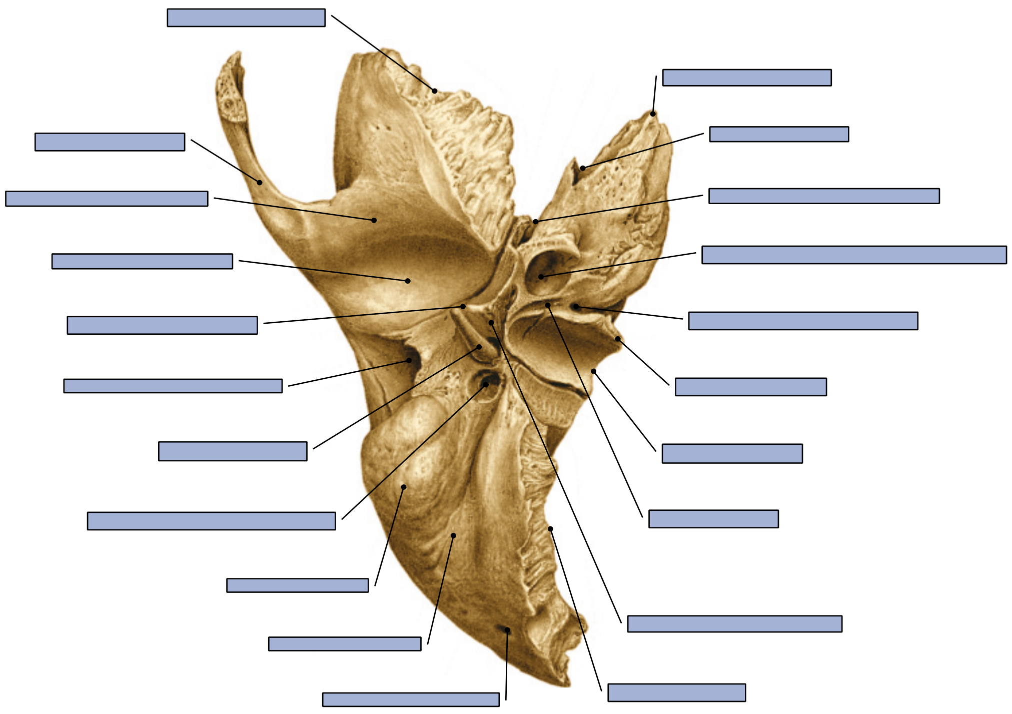


Figure 13.1: Medical drawing, source: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23.Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München. Labeled by our approach (variant TSCH).

Besides readability, also the aesthetics of the figures including their labelings play a central role in professional books. Each figure and its labels are subject to book-specific design rules. Our approach stands out by its ability to support an easy integration of these specific design rules. It particularly relies on only a few key assumptions that most figures with external label placement have in common. Other constraints and rules can easily be patched in according to demand.

To validate our approach, we were in contact with both a layout artist and two editors of the human anatomy atlas Sobotta. Both the layout artist and the two editors stated that label placement is a mechanical, but extensively time-consuming task that is mainly done by hand. The tool support basically comprises simple operations such as placing text boxes and drawing line segments. Based on medical drawings annotated by the authors of the atlas for human anatomy, the layout artist creates the layout of each double page of the book. Using the annotated information, this includes arranging explanatory texts, figures, and labels around the figures. The interviewed layout artist stated that he needs about two hours to create the layout of a double page. Hereby, he spends a large portion of his time on label placement.

Hence, with around 1200 figures in a single volume better tool support would clearly help in improving the process of creating such books. Further, with the upcoming applications on mobile devices, figures are deployed in differently scaled settings, which requires different external labelings for the same figure. Then, at the latest, automatic approaches become inevitable.

Contribution and Outline. Our approach bridges the gap between practical and theoretical results on external labeling; see Chapter 10 for a detailed discussion on related work. Like many of the theoretical results, it uses a clearly and mathematically defined model to guarantee pre-defined design rules. However, in contrast to preceding research, our approach is significantly more flexible. After introducing some terminology (Section 13.2), we present a list of drawing criteria for external label placement that we have extracted from interviews with domain experts as well as a semi-automatic analysis of 202 figures printed in the Sobotta [PW13] atlas (Section 13.3).

Based on a reasonable subset of the most important criteria, we introduce a flexible formal model for *contour labeling*, which is a generalization of boundary labeling (Section 13.4). Afterwards, we describe a basic dynamic programming approach that solves the mathematical problem optimally (Section 13.5). Our approach allows to include further drawing criteria both as *hard* and *soft* constraints, where hard constraints must not be violated at all and the compliance of soft constraints is rated by a cost function. Previous works rarely use hard constraints or cannot easily include new hard constraints. Moreover, in contrast to previous work, our approach also takes consecutively placed labels into account. At first glance this seems to be a small improvement, but in fact it is important to obtain an appealing labeling where, for example, labels have regular distances or the angles of consecutive labels should be similar. Further, our approach supports labels of different size. Indeed, for each point feature, the user can pre-define a set of different label shapes, which do not need to be rectangles. This may be used to model different ways of text formatting supporting single and multi-line labels. Further, the user may specify for each label an individual set of candidate positions that is used for the label placement procedure. Moreover, the user may mark areas that are not allowed to be overlapped by labels including their leaders. This is important to avoid undesired overlaps with the figure or to integrate the figure along with its labeling into a double page with explanatory text. The approach also allows to pre-define groups of labels that are placed consecutively, which is required when naming features that are semantically related.

Our approach is not limited to the described features, but other criteria can be incorporated easily. The strength of our approach comes at the cost of a high asymptotic running time of $O(n^8)$, where n describes the complexity of the input instance. Recently, Keil et al. [Kei+17] presented a similar general dynamic programming approach for computing an independent set in outerstring graphs, which can be utilized

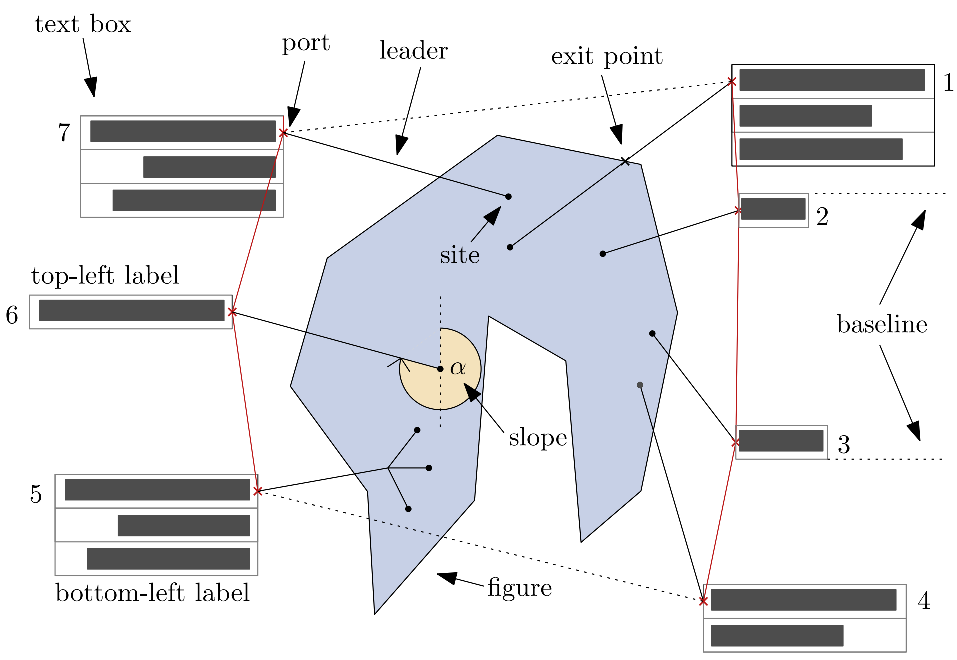


Figure 13.2: Terminology. The radial ordering is given by the numbers.

to solve contour labeling in $O(n^6)$ time for a general cost function rating individual labels (see Chapter 12 for more details). However, it cannot take joint costs of two consecutive labels into account. Similarly, Fink et al. [Fin+12] only consider single labels in their cost functions. In contrast to Fink and Suri [FS16], our approach is significantly faster ($O(n^8)$ instead of $O(n^{15})$) and it supports non-uniform labels and more general shapes for the figure's contour. With some engineering (Section 13.6), we can solve realistically sized instances in adequate time obtaining high layout quality as is shown in our evaluation (Section 13.7) on a large benchmark set of real-world instances.

13.2 Terminology

For the purpose of this chapter, we slightly extend and adapt the terminology introduced in Chapter 10. In particular, for the convenience of the reader, we repeat already defined terms that we specifically use throughout this chapter.

An *illustration* with external labeling consists of a *figure* as well as a set of labels outside of the figure naming single point features of the figure. Hereby a *label* consists of a *text box* that lies outside the figure and a line segment connecting the text box with its point feature; see Figure 13.2. We call the line segment the *leader* and the point feature the *site* of the label. More precisely, the text box is a rectangle containing a (possibly multiline) text. Typically, the rectangle is not displayed, but it is used for the further description. We assume that the leader of a label ends on the boundary of the text box; we call that point the *port* of the label. A leader is directed from its site to its port. A label whose leader goes to the left is called a *left label* and a label whose leader goes to the right is called a *right label.* Analogously, a label is a *top label* (*bottom label*) if its leader goes upwards (downwards). The *baseline* of a bottom-right label is the horizontal half line that emanates from the bottom-right corner of the label's text box to the right. For a top-right label the *baseline* is the horizontal half line that emanates from the top-right corner of the label's text box to the right. The baselines for bottom-left and top-left labels is defined symmetrically. We define the *slope* of a leader as the clockwise angle (starting at 12 o'clock) between the leader and the vertical segment going through the connected site. A leader of a labeling intersects the contour of the figure at its *exit point*; in case that a leader intersects a figure multiple times, we regard the intersection point closest to the port. Traversing the figure's boundary in clockwise order starting from the boundary's topmost point defines an ordering on the exit points of the leaders and accordingly on the labels; we call this the *radial ordering* of the labeling. Two labels are *consecutive* if one directly follows the other in the radial ordering. The *labeling contour* is the polygon that connects the ports of the labels in the given radial ordering.

13.3 Drawing Criteria

Based on interviews with domain experts (one layout artist and two editors of [PW13]) and a semi-automatic analysis of handmade medical drawings with external labeling, we extracted the following set of important layout quality criteria; see also Section 13.7 for more details.

Drawing criteria for sites.

S1 *Shape.* Sites are represented by small points in the drawing.

S2 *Position.* The position of a site is prescribed by domain experts and can be assumed to be fixed and given.

S3 *Type.* Either a site has its own label, or multiple sites have the same label. In the latter case the leaders are bundled forking at a certain point; see Label 5 in Figure 13.2.

For simplicity, we assume that we only have sites of the first type, but with some engineering our algorithms can also be adapted to the second case. Now consider an external labeling of a medical drawing. We have extracted the following criteria.

General drawing criteria.

G1 *Externality.* The text boxes of the labels are placed outside the drawing in the available areas.

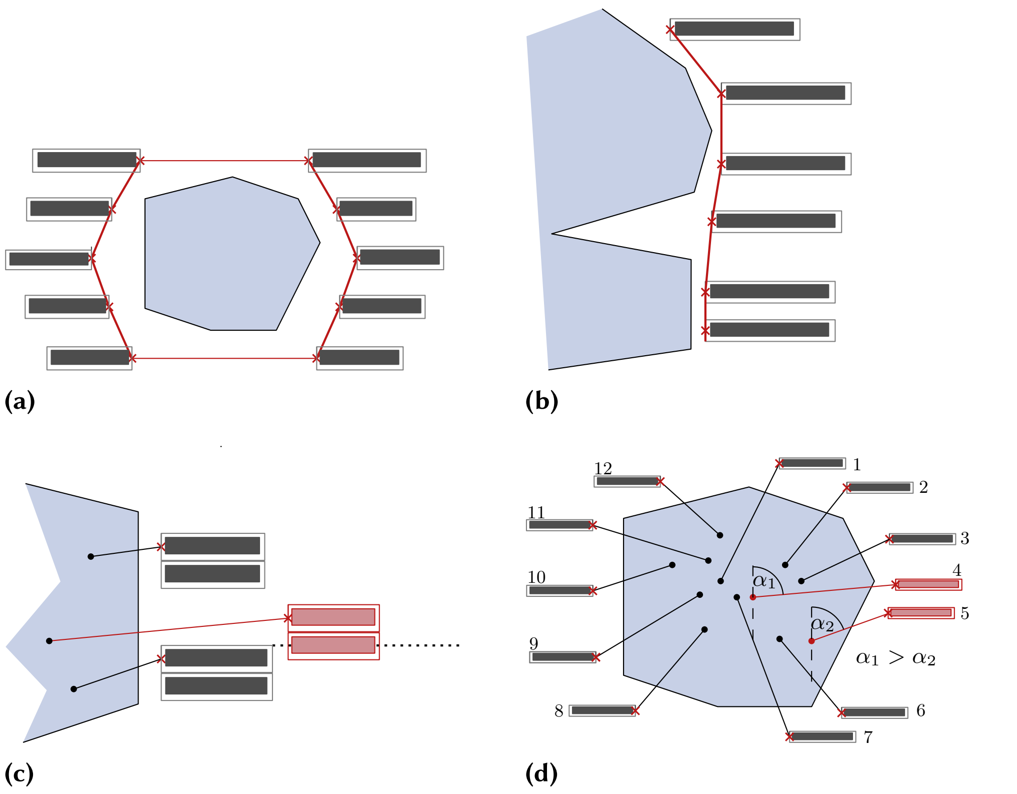


Figure 13.3: Drawing criteria. (a) Criterion G3 (b) Criterion G5 (c) Criterion T5, which is violated by the red middle label. (d) Criterion T4, which is violated by Label 4 and Label 5.

G2 *Planarity.* To sustain readability labels may not overlap or intersect each other.

G3 *Simple shape.* The labeling contour should be simple avoiding turning points; see Figure 13.3(a).

G4 *Left/right side.* The radial order of a labeling can be partitioned into a sequence of left labels and a sequence of right labels. Consequently, the labeling contour can be partitioned into a *left labeling contour* and a *right labeling contour.*

G5 *Similarity.* The labeling contour *mimics* the contour of the figure such that small "indentations" of the figure are not taken into account; see Figure 13.3(b).

G6 *Grouping.* Labels may be required by the designer to appear consecutively in the radial ordering of the labeling.

Drawing criteria for text boxes.

T1 *Spacing.* The vertical distances between text boxes are preferably uniform. Distances less than the height of one text line should be avoided, but may be admissible if not preventable.

T2 *Appearance.* For all labels the same font is applied. Differences in the importance of labels may be expressed by different emphasis (bold, italic).

T3 *Single/Multi line.* If possible, a text box should consist of a single-line text. Only due to the available space, text boxes may consist of multiple lines.

T4 *Ports.* For left (right) labels the port lies on the right (left) edge of the text box in the vertical center of the first text line.

T5 *Staircase.* Let ℓ_1 and ℓ_2 be two consecutive labels. Neither ℓ_1 nor ℓ_2 intersects the baseline of the other; see Figure 13.3(c).

Drawing criteria for leaders.

L1 *Length.* The part of a leader covering the figure should be (preferably) short.

L2 *Distinctiveness.* Leaders running close together should not be parallel to avoid reader confusion.

L3 *Distance.* Leaders preferably comply with a minimum distance to sites of other leaders.

L4 *Monotonicity.* The slope of the leaders increases with respect to the radial ordering of the leaders; see Figure13.3(d).

Typically a labeling does not fully satisfy all these criteria, but criteria may contradict each other requiring appropriate comprises. For example requiring monotonicity (L4) may enlarge the total leader length, which conflicts with criterion L1. Our approach is characterized by the fact that these compromises are not already made during the design of the labeling algorithm, but they lie in the hand of the layout artist applying the algorithm. Specifically, our approach only needs criteria S2, G1, G2, G4 and T5 as hard constraints not to be violated. Further, we assume that we are given a simple shape (G3) enclosing the figure and prescribing possible positions of ports. All other criteria are optional, but can be easily patched in as either hard or soft constraints as needed, which allows interactive and semi-automatic approaches. Hereby the compliance of soft constraints is rated by means of a general cost function that can be defined when applying the algorithm. In our interviews the domain experts strongly emphasized the importance of G2 and G3. They further pointed out that labels should not be placed behind other labels, which we express by T5. We further analyzed 202 medical drawings of [PW13] in a semi-automatic way. All of these examples satisfy S1, G1, G2, G4, and T4. Further, 18 figures contain at least one set of labels that are explicitly grouped by a large curly brace (G6). Only a dwindling small percentage (0.4%) of all labels violate the staircase property (T5) and about 6.2% violate monotonicity (L4). Since the other criteria are soft, we did not quantitatively check these in the semi-automatic analysis; yet, they are well founded in the conducted interviews.

13.4 Formal Model

We now describe a formal model for external label placement. As input we are given a simple polygon F that describes the contour of the figure and contains n sites to be labeled. We denote the set of the sites by S and assume that the sites are in general position, i.e., no three sites are colinear[1]. For each site $s \in S$ we describe its *label*[2] ℓ by a rectangle r and an oriented line segment λ that starts at s and ends on the boundary of r. We call λ the *leader* of ℓ, r the *text box* of ℓ, and the end point of λ on r the *port* of ℓ. The other end point is the site of ℓ; see Figure 13.2. In the following, we only consider labels whose text boxes satisfy T4.

A set $\mathcal{L}$ of labels over S is called an *external labeling* of (F, S), if (1) $|\mathcal{L}| = |S|$, (2) for each site $s \in S$ there is exactly one label in $\mathcal{L}$ that belongs to s, and (3) every text box of a label in $\mathcal{L}$ lies outside of F. If no two labels in $\mathcal{L}$ intersect each other, $\mathcal{L}$ is *planar*. A labeling $\mathcal{L}$ is called a *staircase labeling* if it satisfies criterion T5.

Let $\mathcal{L}$ be a planar labeling. Let $\ell_1, \ldots, \ell_n$ be the labels of $\mathcal{L}$ in the radial ordering. For simplicity we define $\ell_{n+1} := \ell_1$. The *cost* c of a labeling $\mathcal{L}$ is defined as $c(\mathcal{L}) = \sum_{i=1}^{n} c_1(\ell_i) + c_2(\ell_i, \ell_{i+1})$, where c_1 is a function assigning a cost to a single label ℓ_i and c_2 is a function assigning a cost to two consecutive labels ℓ_i and ℓ_{i+1}. We note that in contrast to previous research the cost function also supports rating two consecutive labels, which is crucial to set labels in relation with each other. Given the cost function c, the problem ExternalLabeling then asks for a planar labeling $\mathcal{L}$ of (F, S) that has minimum cost with respect to c, i.e., for any other planar labeling $\mathcal{L}'$ of (F, S) it holds $c(\mathcal{L}) \leq c(\mathcal{L}')$.

We consider the special case that the ports of the labels lie on a common *contour* enclosing F. In contrast to classical boundary labeling, which assumes a rectangular figure, this contour schematizes the shape of the figure with a certain offset; in Section 13.7 we shortly describe how to construct a reasonable contour. Thus, the contour describes the common silhouette formed by the labels. We assume that the contour is given as a simple polygon C enclosing F. An external labeling $\mathcal{L}$ is called a *contour labeling* if for every label of $\mathcal{L}$ its leader lies inside C and its port lies on the boundary ∂C of C. Since not every part of C's boundary may be suitable for the placement of labels, we require that the ports of the labels are contained in a given subset $P \subseteq \partial C$ of candidate ports. If P is finite, the input instance has *fixed ports* and otherwise *sliding ports*.

Observation 13.1. *In a planar contour labeling the ports of the labels induce the same radial ordering with respect to C as the exit points of the labels with respect to F.*

A tuple $I = (C, S, P)$ is called an *instance* of contour labeling. The *region* of I is the

[1]This assumption can be met by slightly perturbing sites.

[2]To ease presentation we define that the leader is a component of the label. In preceding research and in Chapter 10 only the rectangle r is called label.

region enclosed by C. We restrict ourselves to convex contours and clearly separated sites and text boxes as follows, implementing Criteria G1 and G3, respectively.

Assumption 13.1. *The contour C is convex and no text box of any label intersects the convex hull of S.*

For all of the 202 analyzed medical drawings it holds that no text box of any label intersects the convex hull of S.

Due to the convexity of C, the contour can be uniquely split into a left and right side described by two maximal y-monotone chains C_{L} and C_{R}, respectively. The following assumption implements Criterion G4.

Assumption 13.2. *A left label has its port on C_{L} and a right label has its port on C_{R}.*

Given a cost function c, the problem ContourLabeling then asks for an (cost) optimal, planar staircase contour labeling $\mathcal{L}$ of (C, S, P) with respect to c, i.e., for any other planar staircase contour labeling $\mathcal{L}'$ of (C, S, P) it holds that $c(\mathcal{L}) \leq c(\mathcal{L}')$.

13.5 Algorithmic Core

In this section we describe how to construct the optimal labeling $\mathcal{L}$ of a given instance (C, S, P) with respect to a given cost function c. To that end we apply a dynamic programming approach. The basic idea is that any optimal contour labeling can be recursively decomposed into a set of sub-labelings inducing disjoint sub-instances. As we show later, these sub-instances are specially formed; we call them *convex sub-instances*. We further show that any such sub-instance can be described by a constant number of parameters over S and P. Hence, enumerating all choices of these parameters, we can enumerate in polynomial time all possible convex sub-instances that an optimal labeling may consist of. For each such sub-instance we compute the cost of an optimal labeling reusing the results of already computed values of smaller sub-instances. In this way we obtain the value of the optimal labeling for the given instance. Summarizing, our approach consists of four steps.

Step 1. Compute all possible convex sub-instances by enumerating all possible choices defined over S and P.

Step 2. In increasing order of the number of contained sites, compute the optimal cost for each convex sub-instance I. More precisely, to compute the optimal cost of I consider all possibilities how I can be composed of at most two smaller convex sub-instances.

Step 3. Consider all possibilities how the input instance can be described by a convex sub-instance. Among these, take the convex sub-instance with optimal cost.

Step 4. Starting with the resulting sub-instance of Step 3, apply a standard backtracking approach for dynamic programming to construct the corresponding labeling with optimal costs.

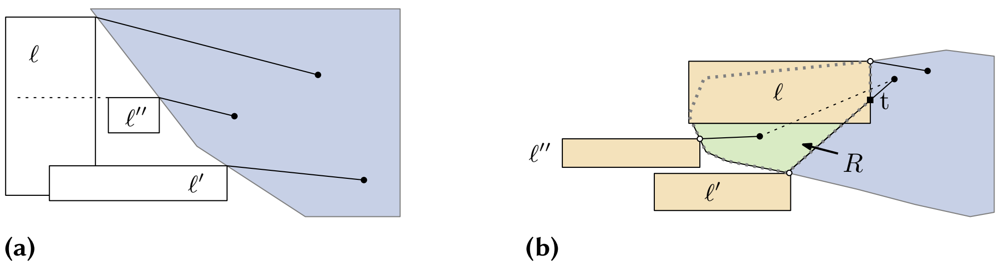


Figure 13.4: Illustration of Lemma 13.1. (a) The base line (dotted line) of ℓ'' intersects the text box of ℓ. (b) The segment connecting the sites of ℓ' and ℓ'' intersects the text box of ℓ.

In the remainder of this section we explain the approach in more detail. In Section 13.5.1, we first prove some structural properties on contour labelings. These properties are crucial for the dynamic programming approach, which we describe in Section 13.5.2.

13.5.1 Structural Properties of Contour Labelings

The intersection of two labels is characterized by three types: the two leaders intersect, the two text boxes intersect or the leader of one label intersects the text box of the other label. The following lemma allows us to find planar labelings by avoiding leader-leader intersections and intersections between two consecutive labels.

Lemma 13.1. *Let I be an instance of* ContourLabeling *and let $\mathcal{L}$ be a staircase contour labeling of I. If no pair of leaders intersect and if no two consecutive labels intersect, then $\mathcal{L}$ is planar.*

Proof. We prove that $\mathcal{L}$ is planar by systematically excluding the possible types of intersections.

Text-box–text-box intersection. Assume that there are two labels ℓ and ℓ' that are not consecutive and whose text boxes intersect. The labels either lie on the same side or on different sides of C.

First consider the case that ℓ and ℓ' belong to different sides; without loss of generality let ℓ be a left label and ℓ' be a right label. Due to T4, text boxes of ℓ and ℓ' may only intersect if the port p of ℓ lies to the right of the port p' of ℓ'. Since p lies on C_L and p' lies on C_R, this contradicts the convexity of C.

Now consider the case that ℓ and ℓ' belong to the same side; without loss of generality let both be left labels; see Figure 13.4(a). For intersecting each other, one of both labels intersects the base line of a left label in between both labels contradicting Criterion T5.

Text-box–leader intersection. Now assume that there is a label ℓ whose text box is intersected by the leader λ' of another label ℓ'; see Figure 13.4(b). We denote the ports

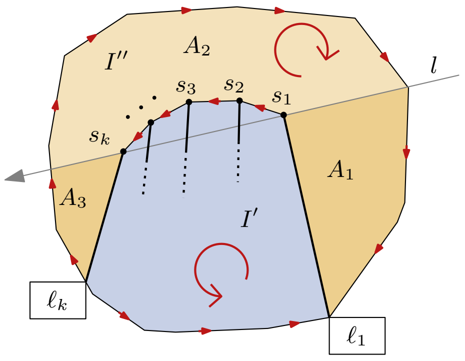


Figure 13.5: Decomposition in convex instance I' (blue region) and concave instance I'' (orange region).

of ℓ and ℓ' by p and p' respectively. Further, let t be the first intersection point of λ' with ℓ going along λ'. We choose ℓ' such that there is no other leader intersecting ℓ's boundary c between p and t. Let R be the region that is bounded by the boundary of ℓ from p to t, the line segment $\overline{tp'}$, and the boundary c' of C from p' to p. Since ℓ and ℓ' are not consecutive, there is a label ℓ'' with port p'' on c'. The site s'' of ℓ'' lies in R because otherwise the leader of ℓ'' intersects c or the segment $\overline{tp'}$. Due to the convexity of C, the segment $\overline{s's''}$ is contained in C. Since s' lies in the complement of R, the segment $\overline{s's''}$ intersects c, which implies that ℓ intersects the convex hull of S contradicting Assumption 13.1. □

In the following, we show that each solution can be subdivided into a finite set of sub-instances of three types. We describe a sub-instance by a simple polygon that consists of two polylines. One polyline is part of the original contour C and the other polyline consists of a convex chain of sites and two leaders. More precisely, assume that we are given a convex chain $K = (s_1, \ldots, s_k)$ of sites with $k \geq 2$ and the two non-intersecting labels ℓ_1 and ℓ_k of s_1 and s_k, respectively; see Figure 13.5. The directed polyline $K' = (p_1, s_1, \ldots, s_k, p_k)$ splits the polygon C into two polygons C' and C'', where p_1 and p_k are the ports of ℓ_1 and ℓ_k, respectively. We consider the sites in the order such that we meet p_1 before p_k when going along the contour of C in clockwise-order starting at the top of C. Further, going along K' we denote the sub-polygon to the left of K' by C' and to the right of K' by C''. With respect to the direction of K', the sub-polygon C' is counter-clockwise oriented, while C'' is clockwise oriented. Further, C'' contains the top point of C. We define that C' contains the sites $s_2, \ldots, s_{k-1}$, while C'' does not contain them.

Thus, the polyline K' partitions the instance (C, S, P) into two sub-instances $I' = (C', S', P')$ and $I'' = (C'', S'', P'')$ such that

(1) $S' \cup S'' = S \setminus \{s_1, s_k\}$ and $P' \cup P'' = P \setminus \{p_1, p_k\}$,
(2) the sites of S' lie in C' or on K and the sites of S'' lie in the interior of C'',
(3) the ports of P' lie on the boundary of C' and the ports of P'' lie on the boundary of C''.

Note that the sites s_1, s_k and the ports p_1, p_k neither belong to I' nor to I'', because they are already used by the fixed labels ℓ_1 and ℓ_k. We call (ℓ_1, ℓ_k, K), which defines the polyline K', the *separator* of C' and C''. For two sub-instances we say that they are *disjoint* if the interiors of their regions are disjoint.

In the following, we only consider sub-instances in which the convex chain K lies to the right of the line l through s_1 and s_k pointing towards s_k from s_1; we will show that these are sufficient for decomposing any instance. Put differently, the chain K is a convex part of the boundary of C' and a concave part of the boundary of C''. We call I' a *convex* sub-instance and I'' a *concave* sub-instance.

The line l splits C'' into three regions A_1, A_2 and A_3; see Figure 13.5. Let A_2 be the region to the right of l and let A_1 and A_3 be the regions to the left of l such that A_1 is adjacent to the leader of ℓ_1 and A_3 is adjacent to the leader of ℓ_k. Depending on the choice of ℓ_1 and ℓ_k, the regions A_1 and A_3 may or may not exist. We call C'' the *exterior* of C' and vice versa. We distinguish the following convex instances.

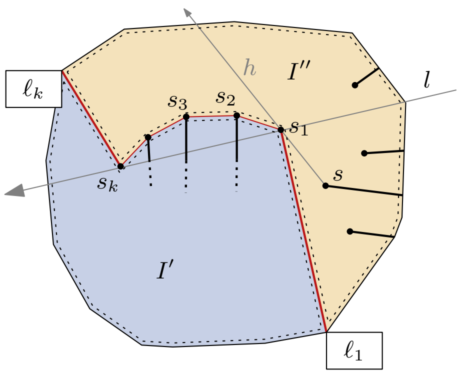
(A) A convex instance has type A if there is a site $s \in A_1$ such that ℓ_1 and the half-line h emanating from s through s_1 separates K from the sites in C''; see Fig 13.6(a).

(B) A convex instance has type B if there is a site $s \in A_3$ such that ℓ_k and the half-line h emanating from s through s_k separates K from the sites in C''; see Fig 13.6(b).

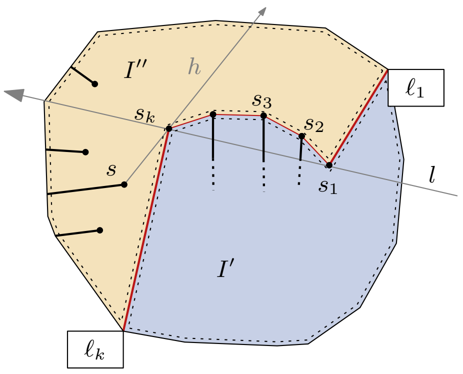
For both types the chain K is uniquely defined by the choice of ℓ_1, ℓ_k and s. Thus, type A and type B instances are uniquely defined by ℓ_1, ℓ_k and s; we denote these instances by $I_\mathrm{A}[\ell_1, \ell_k, s]$ and $I_\mathrm{B}[\ell_1, \ell_k, s]$, respectively. We call s the *support point* of the instance. In case that C'' is empty, the chain K is already uniquely defined by ℓ_1 and ℓ_k and we write $I_\mathrm{A}[\ell_1, \ell_k, \perp]$ and $I_\mathrm{B}[\ell_1, \ell_k, \perp]$. Hence, we can enumerate all such instances by enumerating all possible triples consisting of two labels and one site. Since each label is defined by one port and one site, we obtain $O(|S|^3|P|^2)$ instances in total. Note that instances of type B are symmetric to type A instances.

For $k = 2$ the chain consists of the sites s_1 and s_2 and the support point is superfluous; such an instance is solely defined by the labels ℓ_1 and ℓ_2 of s_1 and s_2, respectively. We call these instances *capstone instances* and denote them by $I_\mathrm{C}[\ell_1, \ell_2]$; see Figure 13.6(c).

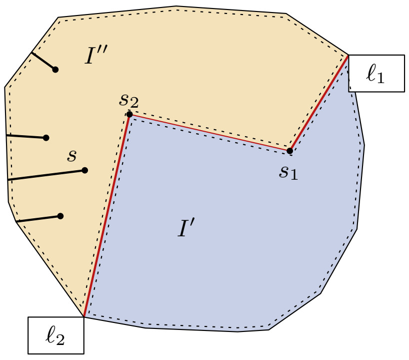
We now show that any labeling can be composed into instances of these three types. We say that an instance is *empty* if its set S of sites is empty. For a labeling $\mathcal{L}$ of an instance I we write $\mathcal{L}|_{I'}$ for the labeling $\mathcal{L}' \subseteq \mathcal{L}$ that is restricted to the sites and ports of the sub-instance I' of I. Let (ℓ_1, ℓ_k, K) denote the separator of I'. For a labeling $\mathcal{L}'$ of a sub-instance I' we require that any leader of any label ℓ in $\mathcal{L}'$ lies in the contour of the sub-instance not intersecting the separator (the sites $s_2, \dots, s_{k-1}$ are excluded from this restriction). In that case we say that the label ℓ is *contained* in I'. We further emphasize that the labels ℓ_1 and ℓ_k are contained in $\mathcal{L}'$, but they are fixed describing the contour of I' so that their sites and ports do not belong to the sites and ports of I', respectively. The next lemma states how to decompose type A and type B instances



(a) Type A instance.



(b) Type B instance.



(c) Capstone instance.

Figure 13.6: Decomposition in convex instance I' (blue) and concave instance I'' (orange). (a) Type A instance with $k > 2$. (b) Type B instance with $k > 2$. (c) Capstone instance.

into smaller type A, type B and capstone instances. Figure 13.7 illustrates Case (i). Case (ii) is symmetric to Case (i).

Lemma 13.2. *Let I be a convex instance with separator $(\ell_1, \ell_k, K = (s_1, \ldots, s_k))$ and $k > 2$. Further, let $\mathcal{L}$ be a planar labeling of I.*

(i) If I has type A, the label $\ell_2 \in \mathcal{L}$ of s_2 splits I into the disjoint sub-instances $I' = I_C[\ell_1, \ell_2]$ and $I'' = I_A[\ell_2, \ell_k, s_1]$ such that any label $\ell \in \mathcal{L}$ is contained in I' or I''.

$$c(\mathcal{L}) = c(\mathcal{L}|_{I_C[\ell_1,\ell_2]}) + c(\mathcal{L}|_{I_A[\ell_2,\ell_k,s_1]}) - c_1(\ell_2)$$

(ii) If I has type B, the label $\ell_{k-1} \in \mathcal{L}$ of s_{k-1} splits I into the two disjoint sub-instances $I' = I_C[\ell_{k-1}, \ell_k]$ and $I'' = I_B[\ell_1, \ell_{k-1}, s_k]$ such that any label $\ell \in \mathcal{L}$ is contained in I' or I''.

$$c(\mathcal{L}) = c(\mathcal{L}|_{I_C[\ell_{k-1},\ell_k]}) + c(\mathcal{L}|_{I_B[\ell_1,\ell_{k-1},s_k]}) - c_1(\ell_{k-1})$$

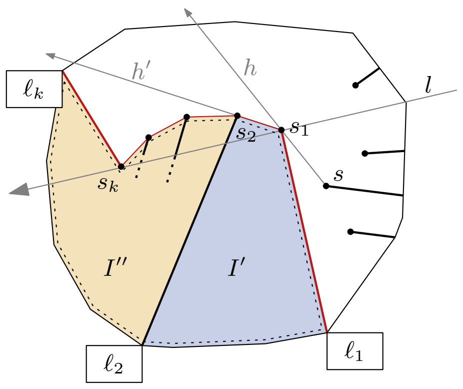


Figure 13.7: Illustration of Lemma 13.2, Case (i).

Proof. We only argue for type A instances. Symmetric arguments hold for type B instances. Consider an arbitrary sub-instance $I = (C, S, P)$ of type A; see Figure 13.7. Since ℓ_2 connects two points of C's boundary, it partitions I into two sub-instances I' and I'' with labelings $\mathcal{L}|_{I'}$ and $\mathcal{L}|_{I''}$ such that any label of $\mathcal{L} \setminus \{\ell_2\}$ either is contained in I' or I''. Let I' be the instance containing s_1 and I'' the other one. Obviously, I' forms the capstone instance $I_C[\ell_1, \ell_2]$. We now show that I'' forms instance $I'' = I_A[\ell_2, \ell_k, s_1]$ of type A.

By definition of I the label ℓ_1 and the half-line h emanating from s through s_1 separates the convex chain K of I from the sites in the exterior of I. Because of the convexity of K, the half-line h' emanating from s_1 through s_2 and the label ℓ_2 separate the convex chain $K' = (s_2, \ldots, s_k)$ from the sites in the exterior of I''. Hence, $I'' = I_A[\ell_2, \ell_k, s_1]$ has type A.

The cost of $\mathcal{L}$ is composed of the costs of $\mathcal{L}|_{I'}$ and $\mathcal{L}|_{I''}$: $c(\mathcal{L}) = c(\mathcal{L}|_{I_C[\ell_1,\ell_2]}) + c(\mathcal{L}|_{I_A[\ell_2,\ell_k,s_1]}) - c_1(\ell_2)$. To not count ℓ_2 twice, we subtract $c_1(\ell_2)$. □

The convex sub-instances I' and I'' of the previous lemma contain fewer sites than I. Thus, recursively applying Lemma 13.2 decomposes I into a set of type A and type B instances until all instances are capstone instances. The next lemma states how to decompose capstone instances into smaller type A, type B and capstone instances. Figure 13.8(a) illustrates Case i, Figure 13.8(b) illustrates Case (ii), Figure 13.8(c) illustrates Case (iii), Case (iv) is symmetric to Case (iii) and Figure 13.8(d) illustrates Case(v).

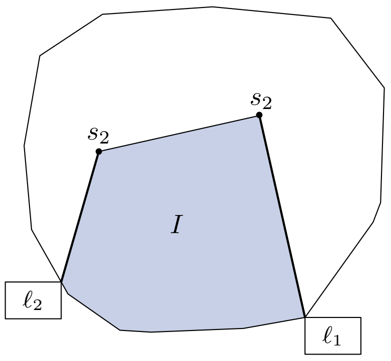
Lemma 13.3. *Let $I = I_C[\ell_1, \ell_2]$ be a capstone instance with separator $(\ell_1, \ell_2, K = (s_1, s_2))$. Further, let $\mathcal{L}$ be a planar labeling of I. One of the following statements applies for I.*

(i) The instance I is empty.

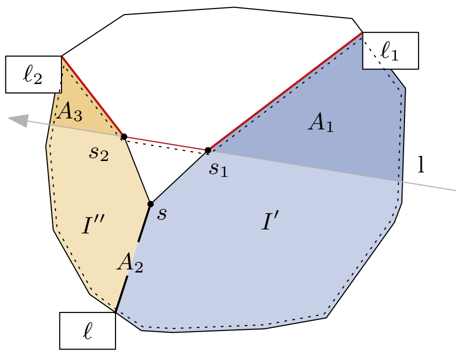
$$c(\mathcal{L}) = c_1(\ell_1) + c_1(\ell_2) + c_2(\ell_1, \ell_2)$$

(ii) There is a label ℓ in $\mathcal{L}$ such that any label in $\mathcal{L}$ is contained in one of the two disjoint capstone instances $I_C[\ell_1, \ell]$ and $I_C[\ell, \ell_2]$.

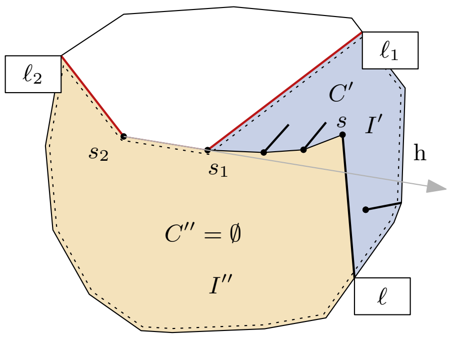
$$c(\mathcal{L}) = c(\mathcal{L}|_{I_C[\ell_1,\ell]}) + c(\mathcal{L}|_{I_C[\ell,\ell_2]}) - c_1(\ell)$$



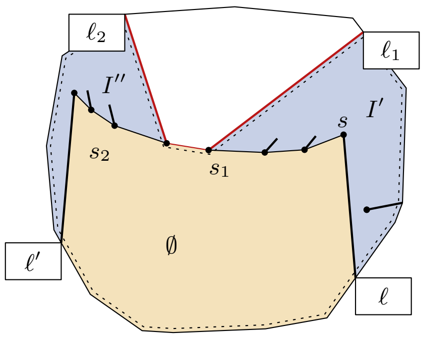
(a) Lemma 13.3, Case (i).



(b) Lemma 13.3, Case (ii).



(c) Lemma 13.3, Case (iii).



(d) Lemma 13.3, Case (v).

Figure 13.8: Decomposition of a convex instance I (dashed polygon). (a) Empty capstone instance. (b) Capstone instance is decomposed into two smaller capstone instances. (c)–(d) Capstone instance is decomposed into type A and type B instances.

(iii) There is a label $\ell \in \mathcal{L}$ s.t. any label in $\mathcal{L}$ is contained in $I_A[\ell_1, \ell, s_2]$.

$$c(\mathcal{L}) = c(\mathcal{L}|_{I_A[\ell_1, \ell, s_2]}) + c_2(\ell, \ell_2)$$

(iv) There is a label $\ell \in \mathcal{L}$ s.t. any label in $\mathcal{L}$ is contained in $I_B[\ell, \ell_2, s_1]$.

$$c(\mathcal{L}) = c(\mathcal{L}|_{I_B[\ell, \ell_2, s_1]}) + c_2(\ell_1, \ell)$$

(v) There are labels $\ell, \ell' \in \mathcal{L}$ with $\ell \neq \ell$ s.t. any label in $\mathcal{L}$ is contained in either $I_A[\ell_1, \ell, s_2]$ or $I_B[\ell', \ell_2, s_1]$.

$$c(\mathcal{L}) = c(\mathcal{L}|_{I_A[\ell_1, \ell, s_2]}) + c(\mathcal{L}|_{I_B[\ell', \ell_2, s_1]}) + c_2(\ell, \ell')$$

Proof. If $I = (C, S, P)$ is empty, the cost of $\mathcal{L}$ are composed by $c(\mathcal{L}) = c_1(\ell_1) + c_2(\ell_2) + c_2(\ell_1, \ell_2)$, which yields Case (i).

So assume that I is not empty. The line l through s_1 and s_2 splits C into three regions A_1, A_2 and A_3; see Figure 13.8(b). Let A_2 be the region to the left of l (going along l) and let A_1 and A_3 be the regions to the right of l such that A_1 and A_3 are adjacent to the leaders of ℓ_1 and ℓ_2, respectively. Further, let p_1 and p_2 be the ports of ℓ_1 and ℓ_2, respectively.

First assume that there is a site s with label $\ell \in \mathcal{L}$ such that the *separating triangle* $\Delta(s_1, s, s_2)$ lies in C and its interior is not intersected by any label in $\mathcal{L}$; the triangle must lie in A_2. Together with ℓ, it partitions I into the two sub-instances I' and I''. They form the capstone instances $I' = I_\mathrm{C}[\ell_1, \ell]$ and $I'' = I_\mathrm{C}[\ell, \ell_2]$ as in Case (ii). It is easy to see that $c(\mathcal{L}) = c(\mathcal{L}|_{I_\mathrm{C}[\ell_1,\ell]}) + c(\mathcal{L}|_{I_\mathrm{C}[\ell,\ell_2]}) - c_1(\ell)$. We subtract $c_1(\ell)$ to not count ℓ twice.

So assume that there is no such separating triangle, which implies that $A_1 \cup A_3$ contains sites. If this were not the case, A_2 would contain a site, because I is not empty. Among these sites we easily could find a site forming a separating triangle. In particular due to Assumption 13.1 such a triangle cannot be intersected by any text box. We distinguish three cases how A_1 and A_3 contain sites.

Case: A_1 contains a site, but A_3 is empty. Let s be a site in A_1. We denote its label by ℓ and the port of ℓ by p. We choose s and ℓ such that no label of any site in A_1 succeeds ℓ in the radial ordering. Let K be the convex chain of sites in A_1 such that K starts at s_1, ends at s, and any site in A_1 lies in the counterclockwise oriented polygon C' defined by p_1, K, p and the part of C in between p and p_1; see Figure 13.8(c). All sites in A_2 also lie in C', because otherwise we could find a separating triangle. Hence, the clockwise oriented polygon C'' defined by p_2, s_2, s_1, K, p and the part of C in between p and s_2 does not contain any site. By the choice of ℓ all labels must be contained in the instance induced by C'. In particular the half-line h emanating from s_2 through s_1 separates K from the sites in the exterior of I'. Hence I' is the type A instance $I_\mathrm{A}[\ell_1, \ell, s_2]$. This yields Case (iii) and $c(\mathcal{L}) = c(\mathcal{L}|_{I_\mathrm{A}[\ell_1,\ell,s_2]}) + c_2(\ell, \ell_2)$.

Case: A_3 contains a site, but A_2 is empty. Symmetrically, we construct a type B instance $I' = I_\mathrm{B}[\ell, \ell_2, s_1]$, which yields Case (iv).

Case: Both A_1 and A_2 contain sites. Combining the construction of the previous two cases, we obtain two labels ℓ and ℓ' that define the type A and type B instances $I' = I_\mathrm{A}[\ell_1, \ell, s_2]$ and $I'' = I_\mathrm{B}[\ell, \ell_2, s_1]$ of case (v); see Figure 13.8(d). □

Lemma 13.2 and Lemma 13.3 describe how to decompose an arbitrary convex sub-instance I into a set of empty capstone instances. The next lemma implies that any labeling of any instance I of ContourLabeling can be decomposed into empty capstone instances.

Lemma 13.4. *Let I be an instance of ContourLabeling and let $\mathcal{L}$ be a planar labeling of I. The first leader ℓ and the last leader ℓ' in the radial ordering of $\mathcal{L}$ define a type A*

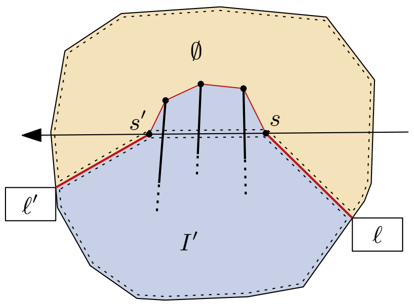


Figure 13.9: Illustration of Lemma 13.4

instance $I' = I_A[\ell, \ell', \perp]$ such that the exterior of I' is empty and

$$c(\mathcal{L}) = c(\mathcal{L}|_{I'}) + c_2(\ell, \ell').$$

Proof. Let $I = (C, S, P)$. Further, let s and s' be the sites and let p and p' be the ports of the two labels ℓ and ℓ', respectively. The polyline (p, s, s', p') partitions C into two sub-polygons; see Figure 13.9. Let C' be the counterclockwise oriented polygon and C'' be the clockwise oriented polygon. Let S'' be the sites in C''. The port of any label in $\mathcal{L}$ lies on the boundary of C', because otherwise ℓ and ℓ' would not be the first and last labels in the radial ordering of $\mathcal{L}$, respectively. Hence, any leader of any label with site in S'' must intersect the line segment $\overline{ss'}$. This in particular implies that any of these sites must lie to the right of the line l that goes through s and s' in that direction. Let H be the convex hull of $S'' \cup \{s, s'\}$. Removing $\overline{ss'}$ from H, we obtain the desired convex chain K, which lies to the right of l. Together with ℓ and ℓ' it forms the type A instance $I_A[\ell, \ell', \perp]$. □

13.5.2 Dynamic Programming

Applying the results of the previous section we present a dynamic programming approach that solves ContourLabeling with fixed ports optimally. For type A, type B and capstone instances the approach creates the three tables T_A, T_B and T_C storing the optimal costs of the considered instances, respectively. We call an instance *valid* if the two labels ℓ_1 and ℓ_k defining the separator do not intersect and comply with T5.

Step 1. We compute all valid instances of type A and type B, and all valid capstone instances.

Step 2. We compute the optimal costs for all convex sub-instances. Let I be the currently considered instance of size $i \geq 0$ with separator $(\ell_1, \ell_k, K = (s_1, \cdots, s_k))$, where the *size* of I is the number of sites contained in I; recall that s_1 and s_k do not belong to I. Considering the instances in non-decreasing order of their sizes, we can assume that we have already computed the optimal costs for all convex instances with size less than i. We distinguish the two main cases $k = 2$ and $k > 2$.

Case k = 2. The instance forms the capstone instance $I_C[\ell_1, \ell_2]$. Let s_1 and s_2 be the sites of ℓ_1 and ℓ_2, respectively. Following Lemma 13.3 we apply five steps.

(1) If $I_C[\ell_1, \ell_2]$ is not empty, we set $w_1 := \infty$ and otherwise

$$w_1 := c_1(\ell_1) + c_1(\ell_2) + c_2(\ell_1, \ell_2).$$

(2) We consider every site s in $I_C[\ell_1, \ell_2]$ such that the *separating triangle* $\Delta(s_1, s_2, s)$ lies in the region of I and does not contain any other site. For all such sites we determine each label candidate ℓ that partitions I into valid disjoint capstone instances $I_C[\ell_1, \ell]$ and $I_C[\ell, \ell_2]$. Let D denote the set of all those labels.

$$w_2 := \min_{\ell \in D} T_C[\ell_1, \ell] + T_C[\ell, \ell_2] - c_1(\ell).$$

(3) We determine every label ℓ of $I_C[\ell_1, \ell_2]$ such that every site of I lies in $I_A[\ell_1, \ell, s_2]$. Let D denote the set of those labels. We set

$$w_3 := \min_{\ell \in D} T_A[\ell_1, \ell, s_2] + c_2(\ell, \ell_2).$$

(4) We determine every label ℓ of $I_C[\ell_1, \ell_2]$ such that every site of I lies in $I_B[\ell, \ell_2, s_1]$. Let D denote the set of those labels.

$$w_4 := \min_{\ell \in D} T_B[\ell, \ell_2, s_1] + c_2(\ell_1, \ell).$$

(5) We determine every pair (ℓ, ℓ') of intersection-free labels of $I_C[\ell_1, \ell_2]$ such that every site of $I_C[\ell_1, \ell_2]$ lies in $I_A[\ell_1, \ell, s_1]$ or $I_B[\ell', \ell_2, s_2]$. Let D denote the set of those pairs labels.

$$w_5 := \min_{(\ell, \ell') \in D} T_A[\ell_1, \ell, s_2] + T_B[\ell, \ell_2, s_1] + c_2(\ell, \ell').$$

In any of the above cases we choose the labels of the candidate set D such that the considered instances are valid. Further, if D is empty in step (i), we set $w_i := \infty$ ($2 \leq i \leq 5$). We set $T_C[\ell_1, \ell_2] := \min_{1 \leq i \leq 5}\{w_i\}$.

Case k > 2. Following Lemma 13.2 we distinguish two sub-cases. If I is a type A instance $I_A[\ell_1, \ell_k, s]$ (where possibly $s = \perp$), we determine every possible label ℓ for site s_2. Let D be the set of those labels. If D is empty, we set $T_A[\ell_1, \ell_k, s] := \infty$ and otherwise

$$T_A[\ell_1, \ell_k, s] := \min_{\ell \in D} T_C[\ell_1, \ell] + T_A[\ell, \ell_k, s_1] - c_1(\ell).$$

If I has type B, we analogously define D for s_{k-1}. If D is empty, we set $T_B[\ell_1, \ell_k, s] := \infty$ and otherwise

$$T_B[\ell_1, \ell_k, s] := \min_{\ell \in D} T_C[\ell, \ell_k] + T_B[\ell_1, \ell, s_{s_k}] - c_1(\ell).$$

In any of the cases above we call the elements in D the *descendants* of the given instance.

Step 3. After computing the optimal costs of all convex instances, we enumerate all possible choices (ℓ, ℓ') of first and last labels in the radial ordering. Let D denote the set of those choices. By Lemma 13.4 any choice $(\ell, \ell') \in D$ forms a convex type A instance $I' = I_A[\ell, \ell', \perp]$. Hence, the optimal cost $\mathrm{OPT}(I)$ of I are

$$\mathrm{OPT}(I) = \min_{(\ell,\ell')\in D} T_A[\ell, \ell', \perp] + c_2(\ell, \ell').$$

Recall that if the convex chain K of I' has length 2, then we have $I' = I_C[\ell, \ell']$ and therefore $T_A[\ell, \ell', \perp] = T_C[\ell, \ell']$. If $\mathrm{OPT}(I) = \infty$, we return that I does not admit a contour labeling.

Step 4. Also storing the according descendants for each instance, we apply a standard backtracking approach for dynamic programming to obtain the corresponding the labeling $\mathcal{L}$.

Theorem 13.1. *For an instance $I = (C, S, P)$ contour labeling, the problem* ContourLabeling *can be solved in $O(|S|^4|P|^4)$ time.*

Proof. We first show the correctness and then argue the running time. Let I be an arbitrary instance of ContourLabeling.

Correctness. Let $\mathcal{L}$ be a labeling constructed by our algorithm. We first show that $\mathcal{L}$ is planar proving the conditions of Lemma 13.1.

By construction we ensure that for a convex instance I the labels of I's separator comply with T5. In particular this implies that any two consecutive labels in $\mathcal{L}$ comply with T5, which implies that $\mathcal{L}$ is a staircase labeling. When computing the descendants D for an instance, we ensure that D does not contain any label that intersects the separator of that instance. By induction this implies that no leader in $\mathcal{L}$ intersects any other leader in $\mathcal{L}$. Further, we ensure that no two consecutive labels intersect. Hence, by Lemma 13.1 the labeling $\mathcal{L}$ is planar.

Finally, we prove that $\mathcal{L}$ is an optimal labeling of I. By Lemma 13.2, 13.3 and 13.4 any labeling can be recursively decomposed into type A, type B, and capstone instances. Our algorithm enumerates all these instances. For each such instance I' we search for the label(s) ℓ (ℓ') that split I' into smaller type A, type B and capstone instances. Since we enumerate all such labels, we also find the label minimizing the cost of I'.

Running Time. We now analyze the running time step by step.

Step 1. We create all possible convex instances of type A and type B and all capstone instances by (conceptually) enumerating all tuples (ℓ_1, ℓ_2) and (ℓ_1, ℓ_2, s), where ℓ_1 and ℓ_2 are labels based on the ports and sites of I and $s \in S$. Since each label is defined by a site and a port, we enumerate $O(|S|^3 \cdot |P|^2)$ tuples. For each instance we also compute the convex chain K and the sites contained in the instance, which needs $O(|S|)$ time assuming that the sites are sorted by their x-coordinates. Sorting the instances by size needs $O(|S|^3 \cdot |P|^2(\log|S| + \log|P|))$ time.

Step 2. Handling a single capstone instance we need $O(|S|^2|P|^2)$ time: For the cases (1)–(4) there are $O(|S| \cdot |P|)$ descendants, while for case (5) there are $O(|S|^2 \cdot |P|^2)$ descendants. Since we consider $O(|S|^2 \cdot |P|^2)$ many instances, we obtain $O(|S|^4|P|^4)$ running time in total for capstone instances.

Handling a single type A or type B instance there are $O(|P|)$ descendants, because the site of the descendants is fixed. Since we consider $O(|S|^3 \cdot |P|^2)$ such instances, we obtain $O(|S|^3|P|^3)$ running time, which is dominated by handling the capstone instances.

Step 3. Enumerating all pairs of first and last labels in the radial ordering can be done in $O(|S|^2|P|^2)$ time.

Step 4. Storing for each instance its descendant of lowest cost, we can do backtracking in linear time.

Altogether, Step 2 dominates with $O(|S|^4|P|^4)$ running time. □

The criteria mentioned in Section 13.3 can be easily patched into the approach. If a criterion should become a hard constraint not to be violated, we simply exclude any sub-instance violating this specific criterion. For example, to enforce monotonicity (L4) we remove any sub-instance whose defining labels violate this criterion. Similarly, if the criterion should become a soft-constraint, we do not exclude the sub-instance, but include its compliance with this criterion into its cost—provided that it can be modeled by the cost functions c_1 or c_2. This is true for all criteria listed in Section 13.3.

13.6 Algorithm Engineering

Initial experiments showed that a naive implementation of the dynamic programming approach does not yield reasonable running times, which matches the high asymptotic running times. In this section, we therefore describe how the approach can be implemented efficiently to prevent these problems for instances of realistic input sizes of $|S| \leq 70$.

13.6.1 Bundling

Instead of considering each possible label individually, we *bundle* labels and redefine the different types of instances based on bundles instead of single labels. More precisely, consider two label candidates ℓ_1 and ℓ_2 of the same site s such that ℓ_1 precedes ℓ_2 in the radial ordering; see Figure 13.10. Let p_1 and p_2 be the ports of ℓ_1 and ℓ_2, respectively. Further, let R be the region enclosed by ℓ_1, ℓ_2 and the part of the contour C that lies between p_1 and p_2 (going in clockwise direction around C). We call ℓ_1 and ℓ_2 *equivalent* if the region R does not contain any site. Hence, ℓ_1 can be *slid* along C to ℓ_2 without

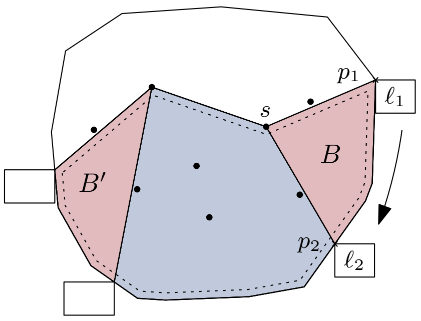


Figure 13.10: Illustration of Bundles.

passing any other site. A *bundle* of s is a set B of labels of s that are pairwise equivalent. A set B is *maximal* if there is no bundle B' with $B \subsetneq B'$. We order the labels according their radial ordering. To ease the description we assume without loss of generality that there is no bundle that contains the top point of C; we can *split* bundles at the top point of C, if necessary.

As presented in Section 13.5.1 we describe a sub-instance $I_A[\ell, \ell', t]$ ($I_B[\ell, \ell', t]$, $I_C[\ell, \ell']$) by two labels ℓ and ℓ' and by a support point t. We now generalize this concept to bundles. For two bundles B and B' of two sites s and s' the *instance set* $\text{Is}_A[B, B', t]$ contains every instance $I_A[\ell, \ell', t]$ with $\ell \in B$ and $\ell' \in B'$. We analogously define the sets Is_B and Is_C. Using these instance sets, we speed up the steps of our algorithm without losing optimality as follows.

Step 1. Instead of creating every instance of every type, we create all possible instance sets Is_A, Is_B and Is_C based on the maximal bundles of the input instance. We note that we do not compute the instance sets explicitly, but each such set Is is uniquely described by the two labels that occur first and last in Is with respect to the radial ordering. Since any two instances of an instance set contain the same sites, we only compute these contained sites once per instance set. We further exclude any instance set that does not permit a labeling at all. To that end, we test whether there is enough space along the enclosed contour for all labels.

Step 2 & Step 3. Instead of computing the optimal cost for each individual instance, we iteratively compute lower and upper bounds of the optimal cost of the instance sets and refine these until we obtain the optimal cost.

More precisely, in each iteration we compute for each instance set Is a lower bound L and an upper bound U for the optimal costs of the instances in Is, i.e., for each instance $I \in \text{Is}$ it holds $L \leq \text{OPT}(I) \leq U$. To that end we define for two bundles B and B' the cost functions c_1 and c_2 as follows.

$$\begin{aligned} c_1(B) &= (\min_{\ell \in B} c_1(\ell), \max_{\ell \in B} c_1(\ell)) \\ c_2(B, B') &= (\min_{\ell \in B, \ell' \in B'} c_2(\ell, \ell'), \max_{\ell \in B, \ell' \in B'} c_2(\ell, \ell')) \end{aligned}$$

Interpreting the result of $c_1(B)$ and $c_2(B, B')$ as two-dimensional vectors we compute

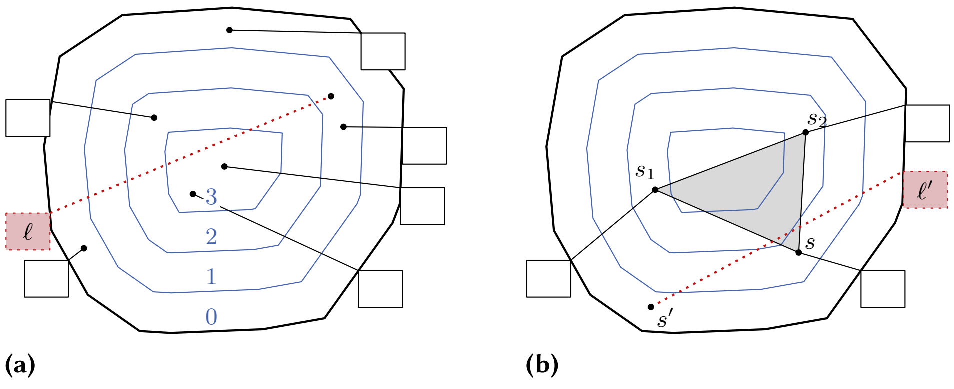


Figure 13.11: Shells (blue). (a) The label ℓ does not directly strive outwards and disturbs the overall appearance of the drawing. (b) No label candidate ℓ' of s' can intersect the separating triangle $\Delta(s_1, s_2, s)$ without intersecting a shell of higher level.

the tables T_A, T_B and T_C in the same manner as before, but this time we use instance sets instead of instances and bundles instead of single labels. We analogously adapt Step 3.

Hence, we obtain for each instance set lower and upper bounds. In particular we obtain a lower bound $\bar{L}$ and an upper bound $\bar{U}$ for the given input instance. In case that $\bar{L} = \infty$, the input instance does not admit a labeling and we can abort the algorithm. Otherwise, we collect all instance sets of Step 3 whose lower bound does not exceed $\bar{U}$ and start a standard backtracking procedure to collect all instance sets whose lower bound does not exceed $\bar{U}$. Any other instance set is omitted, because it cannot be part of the optimal solution. Afterwards, we split each bundle B into two half-sized bundles B_1 and B_2, i.e., let $\ell_1, \dots, \ell_h$ be the labels in B with respect to the radial ordering. We then set $B_1 = \{\ell_1, \dots, \ell_m\}$ and $B_2 = \{\ell_{m+1}, \dots, \ell_h\}$ where $m = \lfloor \frac{h}{2} \rfloor$. Thus, based on those new bundles we obtain new corresponding instance sets. We start the next iteration with the newly created instance sets.

We repeat this procedure until each bundle contains exactly one label. Thus, the bounds $\bar{L}$ and $\bar{U}$ equal the optimal cost of the input instance. Doing backtracking we construct the according labeling.

Shells. In this section, we describe two adaptations that further accelerate the approach. In contrast to the previous section these techniques do not necessarily preserve optimality. In Section 13.7 we show that in practice our heuristics achieve near-optimal solutions in reasonable time.

We observe that leaders typically point outwards without passing through the *center* of the figure; see Figure 13.11(a). This guarantees short leader lengths and leaders fit in the overall appearance of the figure. We use this as follows. Based on the contour C we construct a set of offset polygons in the interior of C such that they have a pre-defined distance to each other; see Figure 13.11(a). We call these polygons *shells*. The level of a shell is the number of shells containing that shell and the level of a site is the level of the innermost shell containing s.

We require for every site that its leader does not intersect a shell with higher level. Hence, we can exclude any label that violates this requirement. The more shells are used the more labels are excluded improving the running time. However, it also becomes more likely that the optimal labeling gets lost. In our experiments we have chosen the shells such that less than 0.9% of the labels in handmade drawings violate this property; see also Section 13.7.

We further speed up Step 2(2). To that end let $I_C[\ell_1, \ell_2]$ be the currently considered capstone instance and let s_1 and s_2 be the sites of ℓ_1 and ℓ_2, respectively. Further, let D be the set of descendants. We only consider descendants that have a level that is at least as high as the level of s_1 and s_2. If such descendants do not exist, we take those with highest level. In case that the shells are convex and nested this particular adaption does not have any impact on the achievable cost, because for any such site s the triangle $\Delta(s_1, s_2, s)$ is contained in the shell of s_1, s_2 or s. It therefore cannot be intersected by any leader of a descendant with a site that has a lower level; see Figure 13.11(b).

Miscellaneous. We further can speed-up the approach as follows.

Simple-Instances. Initial experiments showed that non-capstone instances are more of theoretical interests proving the optimality of the approach, but typically the optimal labeling can be decomposed into capstone instances. Hence, it lends itself to only consider capstone instances; particularly Step 2(3)–(5) are omitted. The asymptotic running time remains the same, because handling capstone instances dominates the running time.

One-Sided-Instances. Assuming criterion G4 we can apply the following speed-up technique preserving the optimality of our approach. Consider a capstone instance $I_C[\ell_1, \ell_2]$ such that both labels ℓ_1 and ℓ_2 are right labels. Let D be the descendants computed in Step 2(2). Indeed we only need to consider the descendants in D with the leftmost site s among all those descendants. It preserves optimality, because no descendant in D of any other site can intersect the separating triangle $\Delta(s_1, s_2, s)$; due to criterion G4 they all lie to the right of the vertical line through s. Here s_1 and s_2 are the sites of ℓ_1 and ℓ_2. Symmetrically, we can apply the same technique for left labels ℓ_1 and ℓ_2 and the descendants with the rightmost site among all descendants in D. The asymptotic running time remains the same, because handling capstone instances with left and right labels dominates the running time.

Small-Triangles. Computing the set D of descendants of a capstone instance $I_C[\ell_1, \ell_2]$ in Step 2(2), any site s in $I_C[\ell_1, \ell_2]$ is considered that forms an empty separating triangle $\Delta(s_1, s_2, s)$. Hereby s_1 and s_2 are the sites of ℓ_1 and ℓ_2. For this speed-up technique we only consider the site s with smallest triangle $\Delta(s_1, s_2, s)$ among those sites reducing the number of descendants in D. This improves the asymptotic running time by a factor of $O(n)$.

13.7 Experimental Evaluation

We have implemented a prototype of our approach incorporating the speed-up techniques of Section 13.6. For simplicity we constructed the contour of the figure based on its convex hull H, i.e., the contour C is an exterior offset polygon of H with distance of 25 pixels. More sophisticated approaches can be applied [Vol+07]. Similarly, the shells are interior offset polygons of the contour having distance of 70 pixels to the next shell. Both choices are ad-hoc values that mimic handmade drawings, but a designer may select them depending on the actual figure. Further, we placed ports by sampling the contour every 10 pixels.

The implemented algorithm uses bundles, the speed-up of one-sided instances, and parallelizes Step 1; see Section 13.6. We distinguish the following variants of our algorithm:

1. Optimal (OPT): No further speed-up techniques.
2. Capstone-Heuristic (CH): Restricted to capstones.
3. Simple-Shell-Heuristic (SCH): Same as CH, but also shells are applied.
4. Triangle-Heuristic (TSCH): Same as SCH including Small-Triangles.

The implementation was done in C++ and compiled with GCC 4.8.5 using optimization level O2. Further, the experiments were performed on an Intel Xeon E5-1630v3 processor clocked at 3.7 GHz, with 128 GB of RAM.

To gain realistic input data, we have vectorized 202 figures of Sobotta [PW13] by extracting the text boxes, the leaders, the sites and the contour F of the figure. In case that a leader λ was connected to multiple sites, we have pragmatically extracted the leader only up to the first fork point p and placed a single site at p. In case that p was not contained F, we took the projection of p on the boundary of F along the half-line from the port of λ through p. The figures contain between 4 and 64 sites; see also Figure 13.12. We used the same data for justifying our drawing criteria.

For reasonably defining the cost functions c_1 and c_2, we first created six labelings for each of five selected figures. These labelings varied in the choice of minimum label distances, as well as enforcing monotonicity (L4) or not. Then we discussed these 30 labelings with a domain expert. She confirmed that monotonicity is an important criterion to obtain balanced labelings and rated the labelings best where labels with less than 30 pixels distance were penalized. Smaller and larger penalty thresholds were rated worse. Further, the expert emphasized that leaders should not run past sites too closely. The analysis of handmade labelings also supports monotonicity: 93.8% of the labels satisfy this property. For about 70% of the test instances violating monotonicity, the violation was less than 10° in maximum. About 93% of these instances have at most 5 violations.

We incorporated these findings into the cost function as follows. Let M be a big constant. Any label candidate and any instance with cost at least M is excluded. In our experiments we set $M = 10^9$. The cost function c_1 takes the leader's length and the smallest distance to sites into account. More precisely, any candidate label whose leader is three times longer than the shortest possible leader for the same site is excluded. Hence, sites located close to the contour of the figure have short leaders, while sites in the center of the figure are not affected by this exclusion at all. Let ℓ be a label candidate and λ be the leader of ℓ. If the distance d of λ to any site (not connected to λ) is less than 10 pixels, we set $c_1(\ell) = \text{length}(\lambda)^2 + M/(100 \cdot d)$ and otherwise $c_1(\ell) = \text{length}(\lambda)^2$. Hence, we penalize both long leaders as well leaders that closely run past a site.

We define c_2 as follows. Let ℓ_1 and ℓ_2 be a pair of possible consecutive label candidates. We set $c_2(\ell_1, \ell_2) = M$ if ℓ_1 and ℓ_2 violate monotonicity by more than 10°, excluding these pairs. For smaller violations we set $c_2(\ell_1, \ell_2) = M/6 + c_v$, which effectively allows at most 5 of these violations in total. If ℓ_1 and ℓ_2 satisfy monotonicity, we set $c_2(\ell_1, \ell_2) = c_v$. Here, c_v is the cost caused by the vertical distance d_v of ℓ_1's and ℓ_2's text boxes: If ℓ_1 and ℓ_2 lie on different sides of C, we set $c_v = 0$. Otherwise, if the vertical distance d_v is less than 5 pixels, we set $c_v = M$ excluding these pairs. If $5 \leq d_v < 30$ pixels, we set $c_v = M/(100 \cdot d_v)$ penalizing too small distances, and in all other cases $c_v = 0$.

We now discuss the quality of the produced labelings as well as the running time of our approaches. Figure 13.14–13.19 at the end of this chapter show the labelings of some of the considered medical drawings.

Quality. To analyze the quality of the labelings constructed by CH, SCH and TSCH, we compare the *cost ratio* $c(\mathcal{L}_A)/c(\mathcal{L}_{\text{OPT}})$, where $\mathcal{L}_{\text{OPT}}$ is created by OPT and $\mathcal{L}_A$ is created by the variant $A \in \{\text{CH}, \text{SCH}, \text{TSCH}\}$; see Figure 13.12. About 73% (CH), 56% (SCH) and 53% (TSCH) of the labelings achieve optimal costs. For 90% of the figures, the algorithms achieve labelings whose costs are at most a factor of 1.06 (CH), 1.75 (SCH) and 1.99 (TSCH) worse than the optimal costs. Only for two figures the optimal costs could not be computed, because their contour is too small to host all labels and no solution exists; hence their original labelings contain nested labels. In such a case the designer would need to adapt the contour. For the majority of the figures monotone labelings (criteria L4) were created. Only for one figure none of the algorithms could create a monotone labeling. For two further figures SCH and TSCH could not create monotone labelings, while the other two approaches did. Finally, for one figure only TSCH did not create a monotone figure.

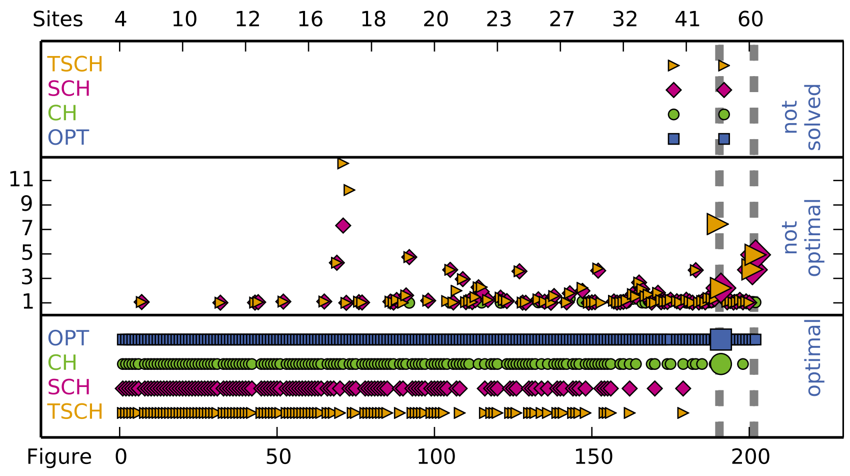


Figure 13.12: Quality. Each column represents one figure broken down into the labelings computed by the algorithm OPT (blue rectangle), CH (green disk), SCH (pink diamond) or TSCH (orange triangle). X-axis: The figures are sorted by their number of sites in increasing order. Y-axis: *not solved*: Labelings that could not be constructed. *not optimal:* Cost ratio of non-optimal labelings. *optimal:* Any labeling $\mathcal{L}_A$ with $c(\mathcal{L}_A) = c(\mathcal{L}_{\mathrm{OPT}})$. Hereby $A \in \{\mathrm{CH}, \mathrm{SCH}, \mathrm{TSCH}, \mathrm{OPT}\}$. Symbols of labelings violating monotonicity (L4) are enlarged and stabbed by a vertical dashed line.

A consulted domain expert stated that the created labelings would be highly useful as initial labeling for the remaining process of laying out the figure. According to the expert, the labelings already have high quality and would require only minor changes, due to aesthetic reasons. These can be hardly expressed as general criteria, but rely on the expertise of the designer. Altogether, the domain expert assessed the approach to be a tool of great use that could reduce the working load of a designer significantly.

Running Time. The average running times of our algorithms range between 7 seconds (TSCH) and 346 seconds (OPT); see Figure 13.12. The variants SCH and TSCH are remarkable faster than OPT; see Figure 13.12. On average they achieve a speedup by a factor of 8.4 and 23.0, respectively. For some figures, TSCH and SCH even achieve a speed up to 198 and 76. Further, TSCH and SCH do not exceed 62 and 312 seconds, respectively. On average TSCH is by a factor 2.5 faster than SCH; in maximum by a factor of 7.1. The variant CH only slightly improves OPT by a factor of 1.4 on average and of 3.7 in maximum.

Since OPT has a high memory consumption, we ran the experiments on a server with 128 GB RAM. When such a system is not available, SCH and TSCH are appropriate alternatives for OPT, because they use significantly less memory, are fast and mostly produce labelings of high quality. To assess the applicability of the approaches in

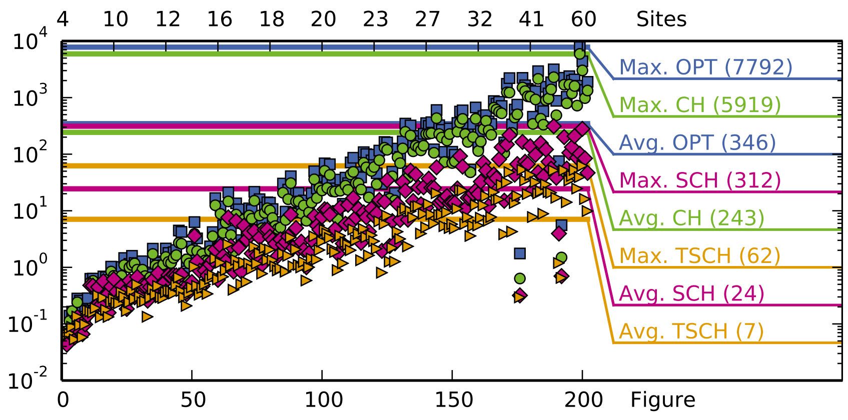


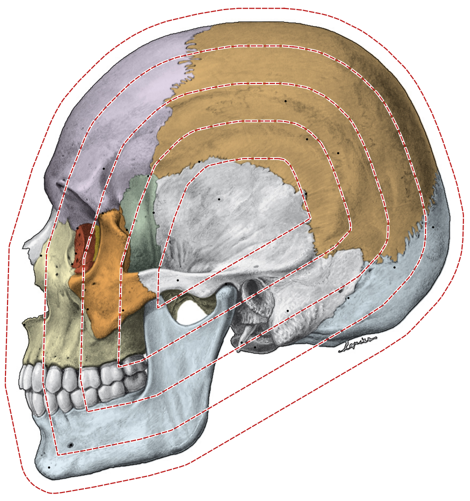
Figure 13.13: Running time in seconds (log. scale). Each column represents one figure broken down into the labelings computed by the algorithm OPT (blue rectangle), CH (green disk), SCH (pink diamond) or TSCH (orange triangle). X-axis: The figures are sorted by their number of sites in increasing order.

a typical setting, we ran both SCH and TSCH on an ordinary laptop with an Intel Core i7-3520M CPU clocked at 2.9 GHz and 8 GB of RAM. In comparison to the previous setting, SCH and TSCH are slower by a factor of 1.24 and 1.22 in maximum, respectively. On average SCH needs 28 seconds and TSCH needs 8 seconds. Within 27 (94) minutes the labelings of all 202 figures were produced by TSCH (SCH); in contrast, one domain expert stated that creating a labeling for a figure with about 50 sites by hand may easily take 30 minutes.

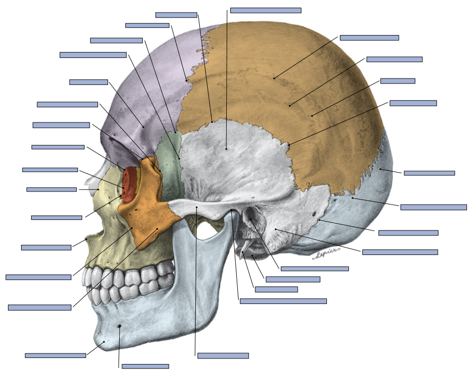
13.8 Conclusions

In this chapter, we presented a flexible model for contour labeling, which we validated through interviews with domain experts and a semi-automatic analysis on handmade labelings. With some engineering, the developed dynamic programming approach can be used to generate labelings of high quality in short time. The presented approach is particularly interesting for creating labelings of large collections of figures that must follow the same design rules; a prominent example are figures in atlases of human anatomy.

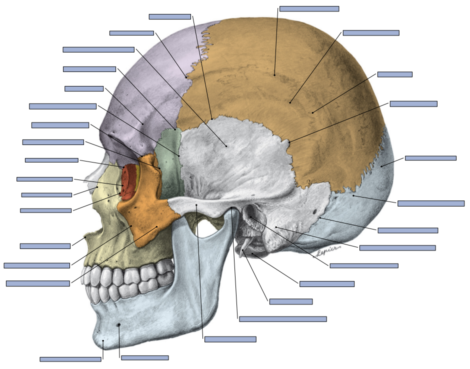
Although the produced labelings already have high quality, we can improve on them by applying post-processing steps, e.g., fix the order of the labels and restart the dynamic programming approach on a larger set of ports to do fine-tuning on the label placement. More sophisticated post-processing steps are future work. Another research direction is to identify more rules for excluding unnecessary instances to further speed up the procedure.



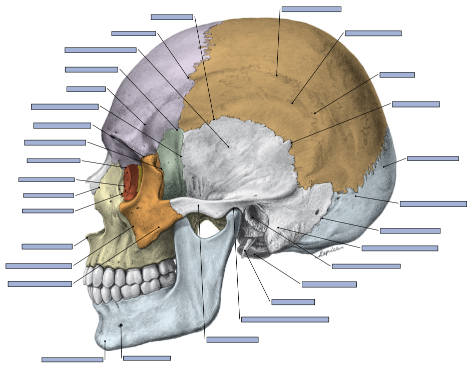
Shells and contour are illustrated by red polygons.



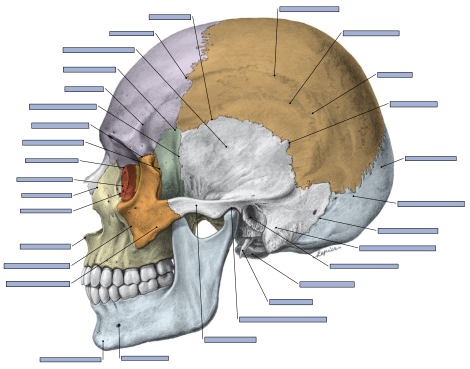
Original Labeling



OPT Labeling (time: 269 sec.)

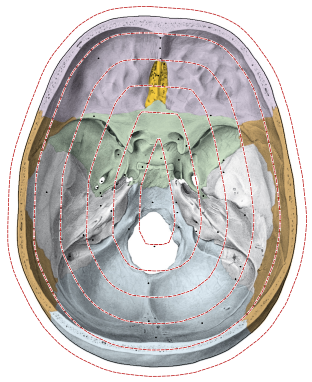


SCH (time: 15 sec., cost ratio: 1.0)

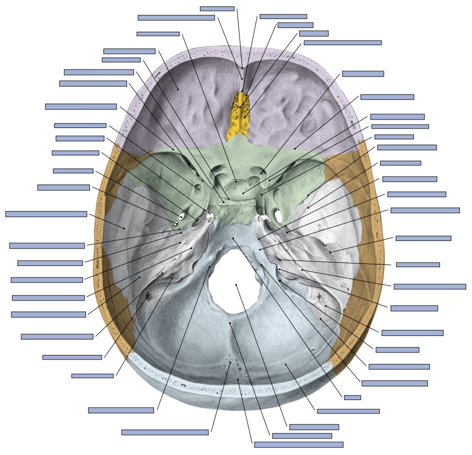


TSCH (time: 9 sec., cost ratio: 1.0)

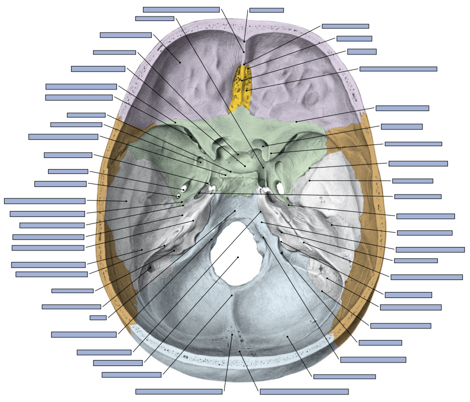
Figure 13.14: Source: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23.Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München; Chapter 8, Figure 6.



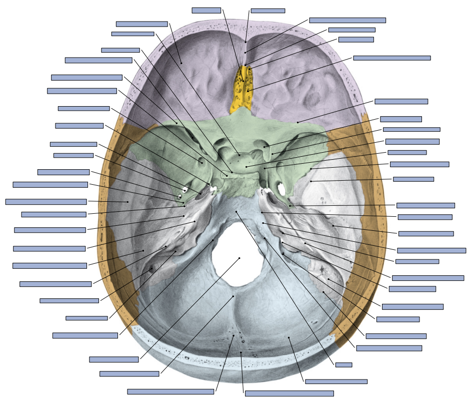
Shells and contour are illustrated by red polygons.



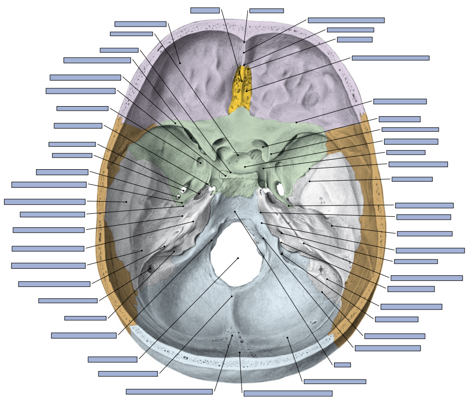
Original Labeling



OPT Labeling (time: 7792 sec.)

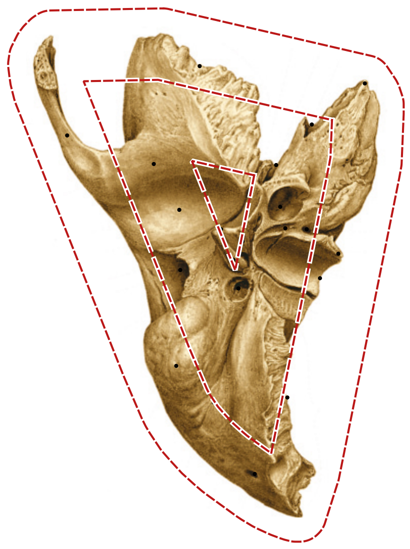


SCH (time: 102 sec., cost ratio: 1.12)

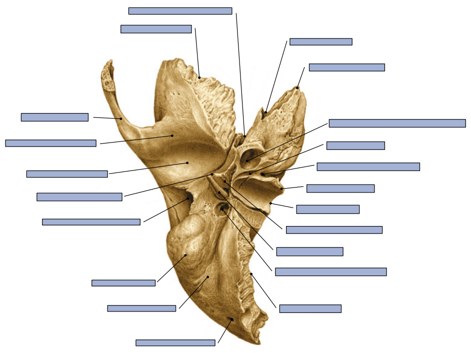


TSCH (time: 39 sec., cost ratio: 1.12)

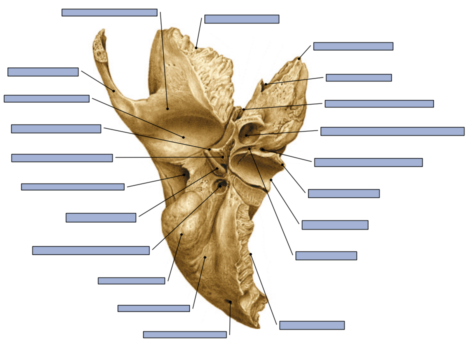
Figure 13.15: Source: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23.Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München; Chapter 8, Figure 12.



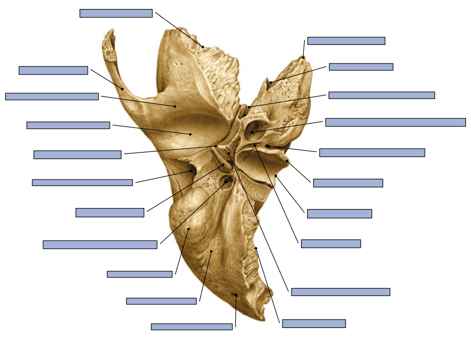
Shells and contour are illustrated by red polygons.



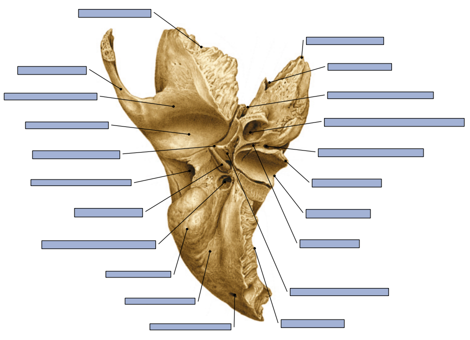
Original Labeling



OPT Labeling (time: 10 sec.)

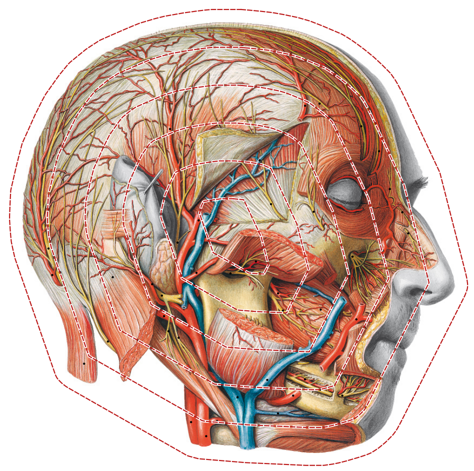


SCH (time: 2 sec., cost ratio: 1.03)

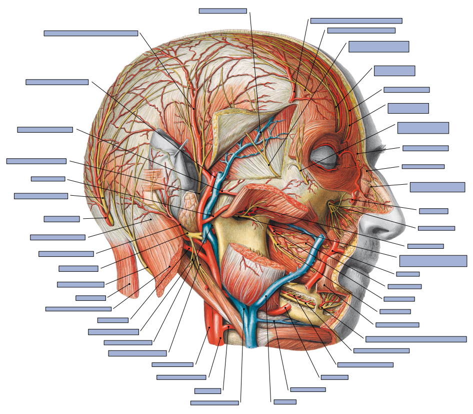


TSCH (time: 1 sec., cost ratio: 1.03)

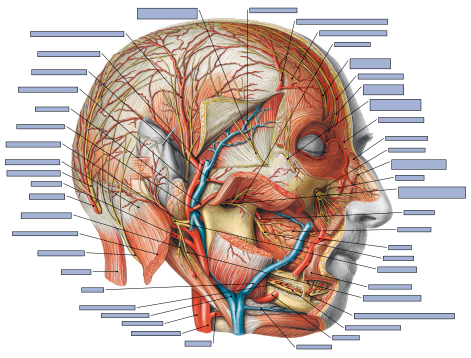
Figure 13.16: Source: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23.Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München; Chapter 8, Figure 44.



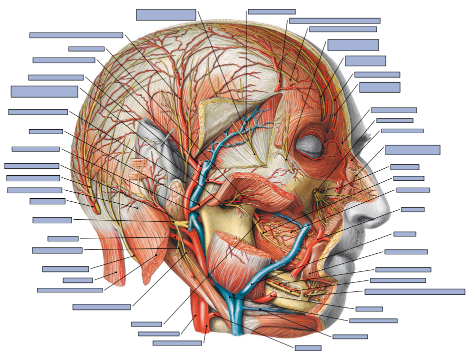
Shells and contour are illustrated by red polygons.



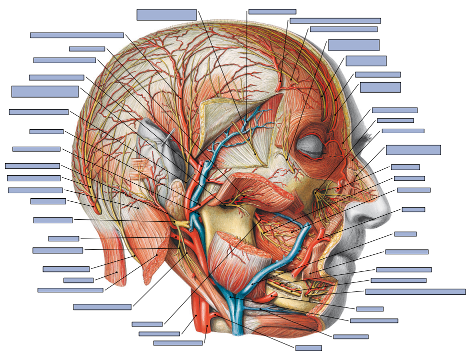
Original Labeling



OPT Labeling (time: 2388 sec.)

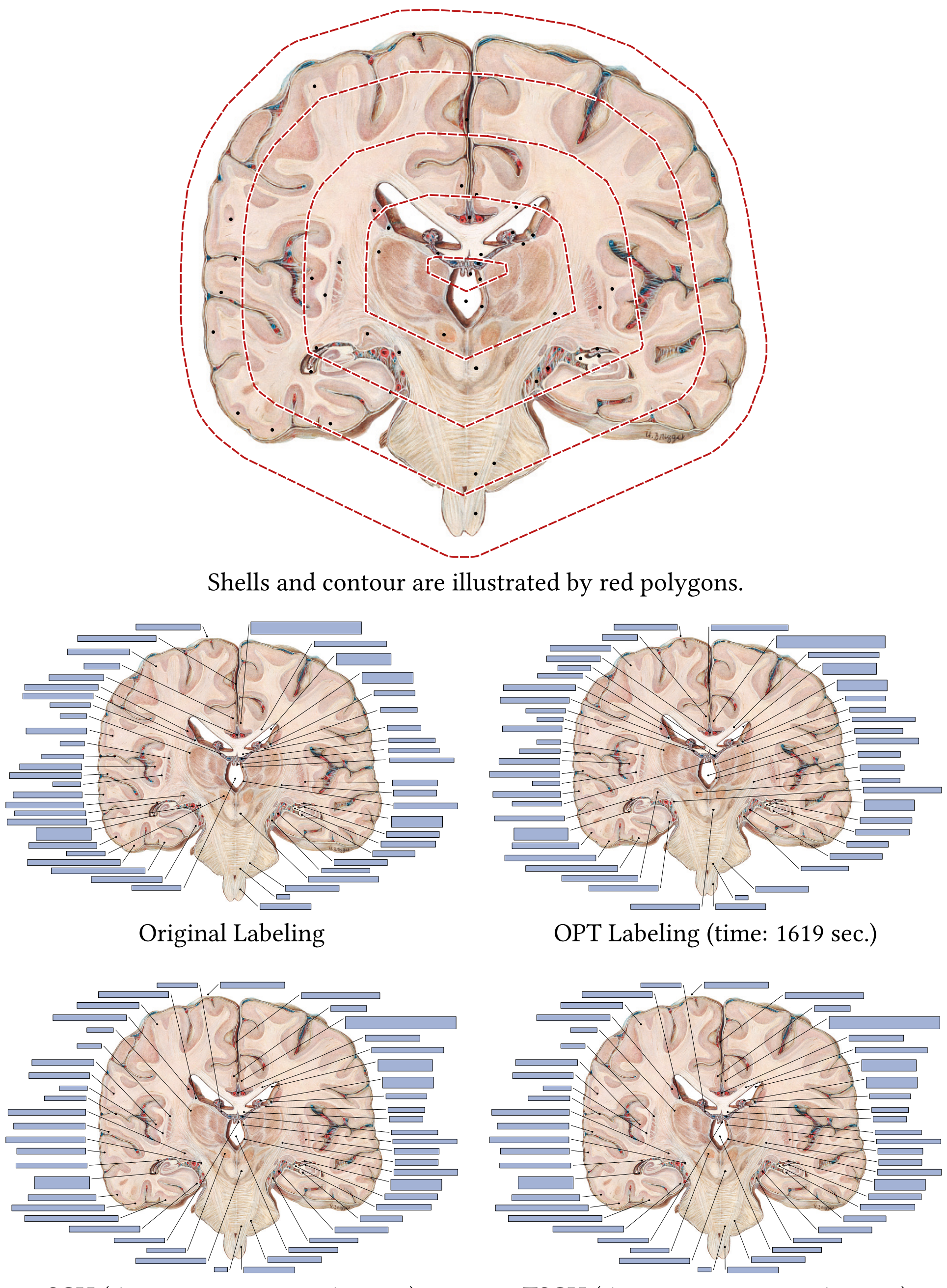


SCH (time: 105 sec., cost ratio: 1.03)



TSCH (time: 34 sec., cost ratio: 1.03)

Figure 13.17: Source: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23.Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München; Chapter 8, Figure 72.



Shells and contour are illustrated by red polygons.

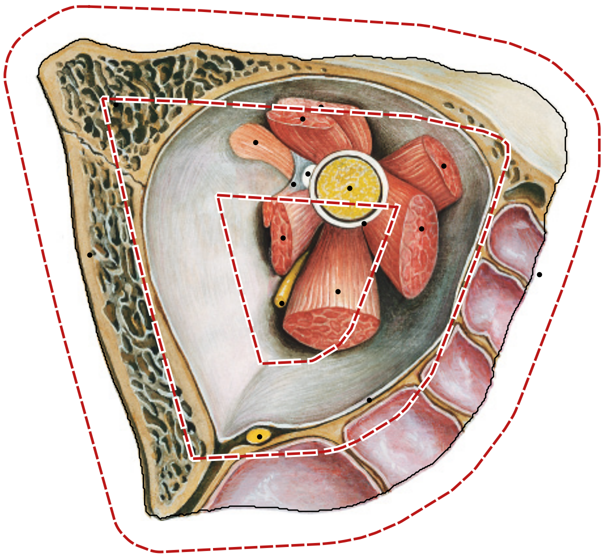
Original Labeling

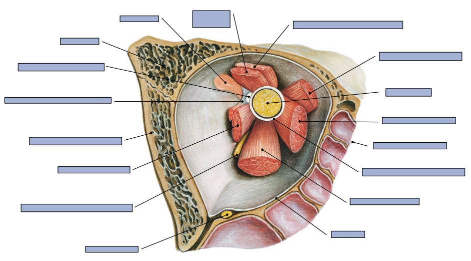
OPT Labeling (time: 1619 sec.)

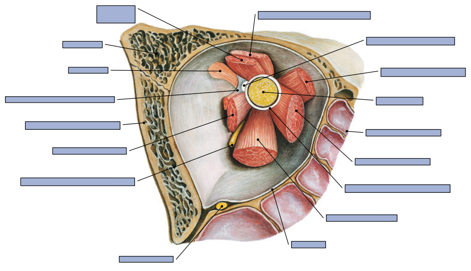
SCH (time: 86 sec., cost ratio: 1.04)

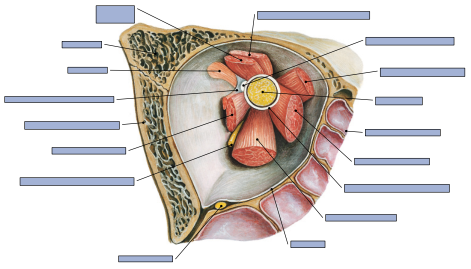
TSCH (time: 25 sec., cost ratio: 1.04)

Figure 13.18: Source: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23.Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München; Chapter 12, Figure 116.


Shells and contour are illustrated by red polygons.


Original Labeling


OPT Labeling (time: 10 sec.)


SCH (time: 2 sec., cost ratio: 1.0)

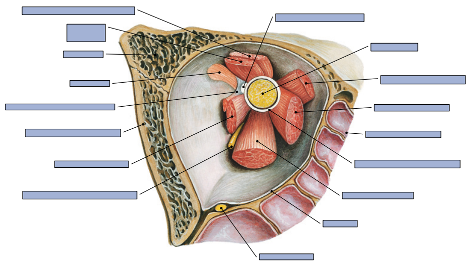

TSCH (time: 1 sec., cost ratio: 10.2)

Figure 13.19: The labeling produced by TSCH is an outlier in our evaluation (see Figure 13.12): Its cost ratio is 10.18. The labeling has high cost because some of the labels' distances are quite small. In a post-processing step this can be corrected. Alternatively, SCH can be used, which achieves optimal cost. Source: Paulsen, Waschke, Sobotta Atlas Anatomie des Menschen, 23.Auflage 2010 ©Elsevier GmbH, Urban & Fischer, München; Chapter 9, Figure 40.

Part III

Conclusions

14 Conclusions

The goal of this thesis was the development of algorithms for label placement in maps and figures. Building on preceding research, we both extended existing and introduced new models for automatic label placement. We validated our models for internal label placement by generally accepted cartographic criteria. For external label placement we conducted a formal user study as well as interviews with domain experts confirming our design decisions. Throughout this thesis, we focused on a precise mathematical formalization of each model, which enabled us to derive mathematically proven results. These results give new insights into the computational complexity bounds of automatic label placement. They comprise proofs on NP-hardness as well as efficient algorithms. In a second step, we utilized the developed algorithms to obtain general workflows for automatic and semi-automatic label placement. We proved their usefulness by their experimental evaluation on real-world data. In the following, we summarize our results for both internal and external label placement, and subsequently give a short outlook on future work.

14.1 Internal Labeling

In the first part of this thesis, we investigated internal labeling for three different cartographic applications, namely label placement in road maps, temporal labeling and label placement in metro maps. We tackled each of these problems using the same systematic approach.

In the first phase of our investigations, we developed formal models for each labeling problem. We made sure that these models are both general and simple such that they can be easily extended. For each setting we also formulated mathematical optimization problems. In contrast to large parts of preceding research, we did not focus on maximizing the number of labels, but we took more complex objectives into account to enforce certain cartographic criteria [Imh75, Chi00]. For label placement in road maps, we argued that maximizing the number of labeled road sections is a desirable objective to increase the informational content of the map. For temporal labeling the overall activity of labels is maximized subject to consistency criteria introduced by Been et al. [BDY06]. For metro maps each stop gets a label candidate assigned such that the constructed labeling blends in with the overall appearance of the map.

In accordance with preceding research on internal label placement, it turned out that the considered optimization problems are NP-hard. We therefore either relaxed the constraints so that the problem is solvable in polynomial time, or we developed efficient approximation algorithms with provable guarantees. More precisely, for road maps we developed an efficient algorithm for labeling road graphs that are trees. For temporal labeling, we introduced constant-factor approximation algorithms for unit square labels and proved that the restricted, yet practically relevant case that no more than k labels can be active at any time can be solved in polynomial-time. For metro maps we presented an efficient algorithm that labels a single metro line optimally under additional assumptions, which are typically satisfied by realistic input instances. These results particularly point out the complexity bounds of the optimization problems.

In the second phase of our investigations, we turned our focus to the practical implementation of our models and approaches. For each of these three problems we embedded the algorithms developed in the first phase into flexible frameworks. To assess their practical usefulness, we experimentally evaluated the frameworks on real-world data. We further compared the results with optimal solutions, which we obtained by means of integer linear programming formulations. For all three problems, we showed that our approaches achieve near-optimal results.

14.2 External Labeling

In the second part of this thesis, we investigated the automatic external label placement in figures. We started with a formal user-study to assess the readability of the four most important leader-types *do*, *po*, *s* and *opo*. While these leader types have been investigated extensively in the scope of boundary labeling, research on their usefulness has been mostly neglected so far. We showed that *s*- and *po*-leaders perform best concerning the assignment of sites with their labels, while *opo*-leaders lag far behind. Conducted interviews indicate that *po*- and *do*-leaders are preferable concerning their aesthetics.

In the subsequent chapters, we focused on external label placement using *po*- and *s*-leaders. For *po*-leaders we applied the boundary labeling model of Bekos et al. [Bek+07] assuming that the labels lie on two or more sides of a given rectangle; in case of two sides the labels lie on two adjacent sides. Based on an intricate analysis on structural properties of *po*-labelings, we introduced an algorithm for the two-sided case that decides the existence of a planar labeling in $O(n^2)$ time. Further, we generalized the algorithm to labels on three and four sides alongside the rectangle.

Finally, inspired by medical drawings in human anatomy, we developed a dynamic programming approach for the external labeling in figures using *s*-leaders. The model is based on few key assumptions that are typically inherent in these kinds of labelings, which we confirmed by a semi-automatic analysis of medical drawings and interviews with domain experts. We presented a versatile and general dynamic programming approach that solves the underlying optimization problem in polynomial time. It stands out by its great flexibility allowing the integration of further criteria as both hard and soft constraints. This provides an automatic enforcement of common design rules into the label placement of large collections of drawings. In contrast to preceding research on boundary labeling, which typically assumes rectangular figures, our approach allows significantly more flexible shapes. Moreover, it provides the rating of consecutive labels, which is mandatory for incorporating label-label-relations such as the distance between two labels. The strength of this approach comes with a high asymptotic running time. We therefore engineered the approach to speed it up. In an experimental evaluation, we proved its practicability for realistically sized instances.

14.3 Outlook

The quality of a labeling—no matter whether internal or external—crucially depends on the applied objective function. As an example take point features in a geographic map: Many algorithms for point features aim at maximizing the number of placed labels. In case of a large number of point features, this may clutter the map and make the actual map content hardly legible. As further consequence, the produced labelings look homogeneous in the sense that the label density is approximately the same in each region. This does not necessarily reflect the actual density of the point features, which, however, may be desirable, e.g., to convey the spatial distribution of a country's population: Densely populated regions contain more labels for cities than sparsely populated regions. Hence, depending on the application, other objectives than maximizing the number of labels might be desirable. Put differently, one subject of research is the development of labeling approaches with more complex objectives. In the best case an algorithm is not specialized in one objective, but it can be adapted to the demands of the user. In this thesis, we already took this direction by focusing on approaches that allow an easy adaption of the objective functions as well as the labeling styles. Since many of the presented approaches rely on dynamic programming, the objective function can easily be exchanged.

Still, the question about appropriate objectives in label placement is not completely solved. This cannot be answered in general, but each application must be considered specifically. This requires a close cooperation between computer scientists and domain experts such as cartographers and designers. Further, user studies are required to validate drawing criteria, e.g., an interesting open research question is the impact of the consistency criteria proclaimed by Been et al. [BDY06]. For the problems considered in this thesis, we therefore also confirmed our models by means of commonly accepted criteria, user studies and interviews with domain experts.

Imhof's classification [Imh75] between point, line and area features was willingly accepted by the computer science community. Heuristics, approximation algorithms, local optimization approaches as well as exact algorithms were developed for each feature type separately. The combination of all features has been almost completely neglected; only few heuristics have been proposed, e.g., by Edmondson et al. [Edm+96]. Since typically many map types contain all three feature types, the question arises, how the labeling approaches for the different map features can be combined. The easiest way is sequentially applying algorithms for the different types of features, e.g., first place all labels for area features, then for all line features, and finally for all point features such that the labels do not overlap. Obviously, this results in labelings in which the first considered feature type is preferred. Thus, considering all three types of map features simultaneously is one of the big open problems in automatic label placement. Especially, when considering different cartographic criteria for each map feature type, the problem becomes intricate and complex.

Similarly, the map content in the background is mostly neglected in algorithm design. Thus, labels may occlude map features, which significantly affects the legibility of the map. Therefore, Imhof required that "names should disturb other map contents as little as possible. Avoid covering, overlapping, and concealment." [Imh75]. Since the labels of line and area features are typically placed inside their features, their placement is less problematic than the placement of labels for point features. Therefore, one prospective research direction in automatic label placement is the incorporation of the map content into the algorithmic design. For external label placement, our presented approach (see Chapter 13) already supports this by excluding or rating label candidates accordingly. Similarly, for road labeling, metro map labeling and temporal labeling the objective functions can be adapted, correspondingly.

For dynamic maps further algorithmic problems need to be investigated. While point features have been considered in the dynamic setting extensively, line and area features got significantly less attention. Hence, the question arise, whether the existing techniques for point features can be adapted to the other two feature types. Furthermore, in the existing models for dynamic map labeling it is typically assumed that the change of the map is known in advance. However, in many interactive scenarios the sequence of map changes cannot be predicted, but relies on the interaction of the user. Hence, reacting to these changes while maintaining certain consistency criteria is a challenging problem, which requires new algorithmic solutions in the area of map labeling. The development of online algorithms with quality guarantees could be one possibility to tackle those interactive scenarios.

Summarizing, much research effort has been invested into automatic label placement in maps and figures. It is not to be expected that a single approach can solve all labeling problems satisfactorily, but depending on the application, specialized models and algorithms are needed. In this thesis we contributed such models and algorithms for four labeling problems bridging the gap between theory and practice.

Bibliography

[AES99] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. **Vertical Decomposition of Shallow Levels in 3-Dimensional Arrangements and Its Applications**. *SIAM Journal on Computing* 29:3 (1999), 912–953.

[AF03] Ronald Azuma and Chris Furmanski. **Evaluating Label Placement for Augmented Reality View Management**. In: *Mixed and Augmented Reality (ISMAR'03)*. IEEE Computer Society, 2003, 66–75.

[AHS05] Kamran Ali, Knut Hartmann, and Thomas Strothotte. **Label Layout for Interactive 3D Illustrations**. *Journal of the WSCG* 13:1 (2005), 1–8.

[AKS98] Pankaj K. Agarwal, Marc van Kreveld, and Subhash Suri. **Label Placement by Maximum Independent Set in Rectangles**. *Computational Geometry: Theory and Applications* 11:3-4 (1998), 209–218.

[APT79] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. **A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas**. *Information Processing Letters* 8:3 (1979), 121–123.

[AW13] Anna Adamaszek and Andreas Wiese. **Approximation Schemes for Maximum Weight Independent Set of Rectangles**. In: *Foundations of Computer Science (FOCS'13)*. IEEE Computer Society, 2013, 400–409.

[Bah+17] Daniel Bahrdt, Michael Becher, Stefan Funke, Filip Krumpe, André Nusser, Martin Seybold, and Sabine Storandt. **Growing Balls in $\mathbb{R}^d$**. In: *Algorithm Engineering and Experiments (ALENEX'17)*. Society for Industrial and Applied Mathematics, 2017, 247–258.

[Bar+15] Lukas Barth, Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. **On the Readability of Boundary Labeling**. In: *Graph Drawing and Network Visualization (GD'15)*. Vol. 9411. Lecture Notes in Computer Science. Springer International Publishing, 2015, 515–527.

[Bar+16] Lukas Barth, Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. **Temporal Map Labeling: A New Unified Framework with Experiments**. In: *Advances in Geographic Information Systems (ACM-GIS'16)*. ACM Press, 2016.

[BDY06] Ken Been, Eli Daiches, and Chee Yap. **Dynamic Map Labeling**. *IEEE Transactions on Visualization and Computer Graphics* 12:5 (2006), 773–780.

[Bee+10] Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. **Optimizing Active Ranges for Consistent Dynamic Map Labeling**. *Computational Geometry: Theory and Applications* 43:3 (2010), 312–328.

[Bek+04] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. **Boundary Labeling: Models and Efficient Algorithms for Rectangular Maps**. In: *Graph Drawing (GD'04)*. Vol. 3383. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2004, 49–59.

[Bek+06a] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. **Multi-Stack Boundary Labeling Problems**. In: *Foundations of Sofware Technology and Theoretical Computer Science (FSTTCS'06)*. Vol. 4337. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2006, 81–92.

[Bek+06b] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. **Polygon Labelling of Minimum Leader Length**. In: *Asia-Pacific Symposium on Information Visualisation (APVis'06)*. Australian Computer Society, Inc., 2006, 15–21.

[Bek+07] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. **Boundary Labeling: Models and Efficient Algorithms for Rectangular Maps**. *Computational Geometry: Theory and Applications* 36:3 (2007), 215–236.

[Bek+08] Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. **Boundary Labeling with Octilinear Leaders**. In: *Algorithm Theory (SWAT'08)*. Vol. 1432. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2008, 234–245.

[Bek+10a] Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. **Boundary Labeling with Octilinear Leaders**. *Algorithmica* 57:3 (2010), 436–461.

[Bek+10b] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. **Area-Feature Boundary Labeling**. *Computer Journal* 53:6 (2010), 827–841.

[Bek+11] Michael A. Bekos, Michael Kaufmann, Dimitrios Papadopoulos, and Antonios Symvonis. **Combining Traditional Map Labeling with Boundary Labeling**. In: *Current Trends in Theory and Practice of Computer Science (SOFSEM'11)*. Vol. 6543. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2011, 111–122.

[Bek+13] Michael Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. **Many-to-One Boundary Labeling with Backbones**. In: *Graph Drawing (GD'13)*. Vol. 8242. Lecture Notes in Computer Science. Springer International Publishing, 2013, 244–255.

[Bek+15] Michael Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. **Many-to-One Boundary Labeling with Backbones**. *Journal of Graph Algorithms and Applications* 19:3 (2015), 779–816.

[Ben+07] Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. **Algorithms for Multi-criteria One-Sided Boundary Labeling**. In: *Graph Drawing (GD'07)*. Vol. 4875. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2007, 243–254.

[Ben+09] Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. **Algorithms for Multi-Criteria Boundary Labeling**. *Journal of Graph Algorithms and Applications* 13:3 (2009), 289–317.

[Ber+08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. **Computational Geometry: Algorithms and Applications**. 3rd ed. Springer-Verlag, 2008.

[BF96] Oliver Bastert and Sándor P. Fekete. **Geometrische Verdrahtungsprobleme**. Tech. rep. 96–247. Universität zu Köln, 1996.

[BG05] Stefan Bruckner and Eduard Gröller. **VolumeShop: An Interactive System for Direct Volume Illustration**. In: *Visualization (Vis'2005)*. IEEE Computer Society, 2005, 671–678.

[BG12] Mark de Berg and Dirk H. P. Gerrits. **Approximation Algorithms for Free-Label Maximization**. *Computational Geometry* 45:4 (2012), 153–168.

[BG13] Mark de Berg and Dirk H. P. Gerrits. **Labeling Moving Points with a Trade-Off between Label Speed and Label Overlap**. In: *Algorithms (ESA'13)*. Vol. 8125. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 373–384.

[BG14] Kevin Buchin and Dirk H. P. Gerrits. **Dynamic Point Labeling is Strongly PSPACE-complete**. *International Journal of Computational Geometry & Applications* 24:4 (2014), 373–395.

[BKS08] Michael A. Bekos, Michael Kaufmann, and Antonios Symvonis. **Efficient Labeling of Collinear Sites**. *Journal of Graph Algorithms and Applications* 12:3 (2008), 357–380.

[BRL09] Enrico Bertini, Maurizio Rigamonti, and Denis Lalanne. **Extended Excentric Labeling**. *Computer Graphics Forum* 28:3 (2009), 927–934.

[BW97] Matthias Bader and Robert Weibel. **Detecting and Resolving Size and Proximity Conflicts in the Generalization of Polygonal Maps**. In: *International Cartographic Conference (ICC'97)*. 1997, 1525–1532.

[C399] Bernard Chazelle and 36 co-authors. **The Computational Geometry Impact Task Force Report**. In: *Advances in Discrete and Computational Geometry*. Vol. 223. American Mathematical Society, 1999, 407–463.

[ČB10] Ladislav Čmolík and Jiří Bittner. **Layout-Aware Optimization for Interactive Labeling of 3D Models**. *Computers & Graphics* 34:4 (2010), 378–387.

[CC09] Parinya Chalermsook and Julia Chuzhoy. **Maximum Independent Set of Rectangles**. In: *Discrete Algorithms (SODA'09)*. Society for Industrial and Applied Mathematics, 2009, 892–901.

[Cha+13] Timothy M. Chan, Hella-Franziska Hoffmann, Stephen Kiazyk, and Anna Lubiw. **Minimum Length Embedding of Planar Graphs at Fixed Vertex Locations**. In: *Graph Drawing (GD'13)*. Vol. 8242. Lecture Notes in Computer Science. Springer International Publishing, 2013, 376–387.

[Cha04] Timothy M. Chan. **A Note on Maximum Independent Sets in Rectangle Intersection Graphs**. *Information Processing Letters* 89:1 (2004), 19–23.

[Chi00] François Chirié. **Automated Name Placement With High Cartographic Quality: City Street Maps**. *Cartography and Geographic Information Science* 27:2 (2000), 101–110.

[CL95] Martin C. Carlisle and Errol L. Lloyd. **On the k-Coloring of Intervals**. *Discrete Applied Mathematics* 59:3 (1995), 225–235.

[CMS94] Jon Christensen, Joe Marks, and Stuart Shieber. **Placing Text Labels on Maps and Diagrams**. In: *Graphics Gems IV*. Academic Press Professional, Inc., 1994, 497–504.

[CMS95] Jon Christensen, Joe Marks, and Stuart Shieber. **An Empirical Study of Algorithms for Point-Feature Label Placement**. *The American Cartographer* 14:3 (1995), 203–232.

[Cor+09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. **Introduction to Algorithms (3. Edition)**. MIT Press, 2009.

[Edm+96] Shawn Edmondson, Jon Christensen, Joe Marks, and Stuart M. Shieber. **A General Cartographic Labelling Algorithm**. *Cartographica* 33:4 (1996), 13–24.

[EKW03] Dietmar Ebner, Gunnar W. Klau, and Rene Weiskirscher. **Force-Based Label Number Maximization**. Tech. rep. Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2003.

[Erl+10] Thomas Erlebach, Torben Hagerup, Klaus Jansen, Moritz Minzlaff, and Alexander Wolff. **Trimming of Graphs, with Application to Point Labeling**. *Theory of Computing Systems* 47:3 (2010), 613–636.

[FA84] Herbert Freeman and John Ahn. **AUTONAP – An Expert System for Automatic Name Placement**. In: *Spatial Data Handling (SDH'84)*. ACM Press, 1984, 544–569.

[Fek99] Jean-Daniel Fekete. **Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualizaition**. In: *Human Factors in Computing Systems (CHI'99)*. ACM Press, 1999, 512–519.

[FG06] Jörg Flum and Martin Grohe. **Parameterized Complexity Theory**. Springer New York, 2006.

[Fin+12] Martin Fink, Jan-Henrik Haunert, André Schulz, Joachim Spoerhase, and Alexander Wolff. **Algorithms for Labeling Focus Regions**. *IEEE Transactions on Visualization and Computer Graphics* 18:12 (2012), 2583–2592.

[Fin+13] Martin Fink, Herman Haverkort, Martin Nöllenburg, Maxwell Roberts, Julian Schuhmann, and Alexander Wolff. **Drawing Metro Maps Using Bézier Curves**. In: *Graph Drawing (GD'13)*. Vol. 7704. Lecture Notes in Computer Science. Springer International Publishing, 2013, 463–474.

[FKS16] Stefan Funke, Filip Krumpe, and Sabine Storandt. **Crushing Disks Efficiently**. In: *Combinatorial Algorithm (IWOCA'16)*. Vol. 9843. Lecture Notes in Computer Science. Springer International Publishing, 2016, 43–54.

[FPT81] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. **Optimal Packing and Covering in the Plane are NP-Complete**. *Information Processing Letters* 12:3 (1981), 133–137.

[FS16] Martin Fink and Subhash Suri. **Boundary Labeling with Obstacles**. In: *Canadian Conference on Computational Geometry (CCCG'16)*. 2016, 86–92.

[Fuc+06] Georg Fuchs, Martin Luboschik, Knut Hartmann, Kamran Ali, Heidrun Schumann, and Thomas Strothotte. **Adaptive Labeling for Interactive Mobile Information Systems**. In: *Information Visualization (InfoVis'06)*. IEEE Computer Society, 2006, 453–459.

[FW91] Michael Formann and Frank Wagner. **A Packing Problem with Applications to Lettering of Maps**. In: *Computational Geometry (SoCG'91)*. ACM Press, 1991, 281–288.

[GHN11] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. **Boundary-labeling Algorithms for Panorama Images**. In: *Advances in Geographic Information Systems (ACM-GIS'11)*. ACM Press, 2011, 289–298.

[GHN15] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. **Multirow Boundary-Labeling Algorithms for Panorama Images**. *ACM Transactions on Spatial Algorithms and Systems* 1:1 (2015), 1–30.

[GHS06] Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. **Agent-based Annotation of Interactive 3D Visualizations**. In: *Smart Graphics (SG'06)*. Vol. 4073. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2006, 24–35.

[GJ79] Michael R. Garey and David S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W. H. Freeman & Co., 1979.

[GKS16] Arthur van Goethem, Marc van Kreveld, and Bettina Speckmann. **Circles in the Water: Towards Island Group Labeling**. In: *Geographic Information Science (GIScience'16)*. Vol. 9927. Lecture Notes in Computer Science. Springer International Publishing, 2016, 293–307.

[GNN13a] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. **Trajectory-Based Dynamic Map Labeling**. In: *European Workshop on Computational Geometry (EuroCG'13)*. Preprint. 2013.

[GNN13b] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. **Trajectory-Based Dynamic Map Labeling**. In: *Algorithms and Computation (ISAAC'13)*. Vol. 8283. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 413–423.

[GNN14] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. **Label Placement in Road Maps**. In: *European Workshop on Computational Geometry (EuroCG'14)*. Preprint. 2014.

[GNN15] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. **Label Placement in Road Maps**. In: *Algorithms and Complexity (CIAC'15)*. Vol. 9079. Lecture Notes in Computer Science. Springer International Publishing, 2015, 221–234.

[GNR11] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. **Sliding Labels for Dynamic Point Labeling**. In: *Canadian Conference on Computational Geometry (CCCG'11)*. 2011, 205–210.

[GNR16a] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. **Consistent Labeling of Rotating Maps**. *Journal of Computational* 7:1 (2016), 308–331.

[GNR16b] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. **Evaluation of Labeling Strategies for Rotating Maps**. *Journal of Experimental Algorithmics* 21:1 (2016), 1–21.

[Goe+13] Arthur van Goethem, Wouter Meulemans, Andreas Reimer, Herman Haverkort, and Bettina Speckmann. **Topologically Safe Curved Schematisation**. *The Cartographic Journal* 50:3 (2013), 276–285.

[Göt+05] Timo Götzelmann, Kamran Ali, Knut Hartmann, and Thomas Strothotte. **Form Follows Function: Aesthetic Interactive Labels**. In: *Computational Aesthetics in Graphics, Visualization and Imaging (CAe'05)*. Eurographics Association, 2005, 193–200.

[Gri+91] Peter Gritzmann, Bojan Mohar, Janos Pach, and Richard Pollack. **Embedding a Planar Triangulation with Vertices at Specified Positions**. *American Mathematical Monthly* 98 (1991), 165–166.

[Har+05] Knut Hartmann, Timo Götzelmann, Kamran Ali, and Thomas Strothotte. **Metrics for Functional and Aesthetic Label Layouts**. In: *Smart Graphics (SG'05)*. Vol. 3638. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2005, 115–126.

[HAS04] Knut Hartmann, Kamran Ali, and Thomas Strothotte. **Floating labels: Applying dynamic potential fields for label layout**. In: *Smart Graphics (SG'04)*. Vol. 3031. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2004, 101–113.

[Hir75] Dan Hirschberg. **A Linear Space Algorithm for Computing Maximal Common Subsequences**. *Communications of the ACM* 18:6 (1975), 341–343.

[Hir82] Stephen Hirsch. **An Algorithm for Automatic Name Placement Around Point Data**. *The American Cartographer* 1:9 (1982), 5–17.

[HM85] Dorit S. Hochbaum and Wolfgang Maass. **Approximation Schemes for Covering and Packing Problems in Image Processing and VLSI**. *Journal of the ACM* 32:1 (1985), 130–136.

[HN15] Jan-Henrik Haunert and Benjamin Niedermann. **An Algorithmic Framework for Labeling Network Maps**. In: *Computing and Combinatorics (COCOON'15)*. Vol. 9198. Lecture Notes in Computer Science. Springer International Publishing, 2015, 689–700.

[HPL14] Zhi-Dong Huang, Sheung-Hung Poon, and Chun-Cheng Lin. **Boundary Labeling with Flexible Label Positions**. In: *WALCOM: Algorithms and Computation (WALCOM'14)*. Vol. 8344. Lecture Notes in Computer Science. Springer-Verlag, 2014, 44–55.

[HTC92] Ju Yuan Hsiao, Chuan Yi Tang, and Ruay Shiung Chang. **An Efficient Algorithm for Finding a Maximum Weight 2-Independent Set on Interval Graphs**. *Information Processing Letters* 43:5 (1992), 229–235.

[Imh62] Eduard Imhof. **Die Anordnung der Namen in der Karte**. *International Yearbook of Cartography* 2 (1962), 93–129.

[Imh75] Eduard Imhof. **Positioning Names on Maps**. *The American Cartographer* 2:2 (1975), 128–144.

[JH15] Yan Jin and Jin-Kao Hao. **General Swap-Based Multiple Neighborhood Tabu Search for the Maximum Independent Set Problem**. *Engineering Applications of Artificial Intelligence* 37 (2015), 20–33.

[Jia+05] Minghui Jiang, Sergey Bereg, Zhongping Qin, and Binhai Zhu. **New Bounds on Map Labeling with Circular Labels**. In: *Algorithms and Computation (ISAAC'04)*. Vol. 3341. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2005, 606–617.

[Kat+09] Bastian Katz, Marcus Krug, Ignaz Rutter, and Alexander Wolff. **Manhattan-Geodesic Embedding of Planar Graphs**. In: *Graph Drawing (GD'09)*. Vol. 5849. Lecture Notes in Computer Science. Springer International Publishing, 2009, 207–218.

[KB08] Jill Kern and Cynthia Brewer. **Automation and the Map Label Placement Problem: A Comparison of Two GIS Implementations of Label Placement**. *Cartographic Perspectives* 60 (2008), 22–45.

[Kei+17] Mark Keil, Joseph Mitchell, Dinabandhu Pradhan, and Martin Vatshelle. **An Algorithm for the Maximum Weight Independent Set Problem on Outerstring Graphs**. *Computational Geometry: Theory and Applications* 60 (2017), 19–25.

[Kin+13a] Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. **Two-Sided Boundary Labeling with Adjacent Sides**. In: *European Workshop on Computational Geometry (EuroCG'13)*. Preprint. 2013.

[Kin+13b] Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. **Two-Sided Boundary Labeling with Adjacent Sides**. In: *Algorithms and Data Structures (WADS'13)*. Vol. 8037. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 463–474.

[Kin+16] Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. **Multi-sided Boundary Labeling**. *Algorithmica* 76:1 (2016), 225–258.

[Kin15] Philipp Kindermann. **Angular Schematization in Graph Drawing**. Dissertation. Universität Würzburg, 2015.

[KLW14] Philipp Kindermann, Fabian Lipp, and Alexander Wolff. **Luatodonotes: Boundary Labeling for Annotations in Texts**. In: *Graph Drawing (GD'14)*. Vol. 8871. Lecture Notes in Computer Science. Springer International Publishing, 2014, 76–88.

[KSW99] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. **Point Labeling with Sliding Labels**. *Computational Geometry: Theory and Applications* 13:1 (1999), 21–47.

[KT98] Konstantinos G. Kakoulis and Ioannis G. Tollis. **A Unified Approach to Labeling Graphical Features**. In: *Computational Geometry (SoCG'98)*. ACM Press, 1998, 347–356.

[Lic82] David Lichtenstein. **Planar Formulae and Their Uses**. *SIAM Journal on Computing* 11:2 (1982), 329–343.

[Lie+95] Thomas M. Liebling, François Margot, D. Müller, Alain Prodon, and L. Stauffer. **Disjoint Paths in the Plane**. *ORSA Journal on Computing* 7:1 (1995), 84–88.

[Lin10] Chun-Cheng Lin. **Crossing-Free Many-to-One Boundary Labeling with Hyperleaders**. In: *Pacific Visualization Symposium (PacificVis'10)*. IEEE Computer Society, 2010, 185–192.

[LKY07] Chun-Cheng Lin, Hao-Jen Kao, and Hsu-Chun Yen. **Many-to-One Boundary Labeling**. In: *Asia-Pacific Symposium on Information Visualisation (APVis'06)*. IEEE Computer Society, 2007, 65–72.

[LKY08] Chun-Cheng Lin, Hao-Jen Kao, and Hsu-Chun Yen. **Many-to-One Boundary Labeling**. *Journal of Graph Algorithms and Applications* 12:3 (2008), 319–356.

[LLP14] Chung-Shou Liao, Chih-Wei Liang, and Sheung-Hung Poon. **Approximation Algorithms on Consistent Dynamic Map Labeling**. In: *Frontiers in Algorithmics (FAW'14)*. Vol. 8497. Lecture Notes in Computer Science. Springer International Publishing, 2014, 170–181.

[LN10] Maarten Löffler and Martin Nöllenburg. **Shooting Bricks with Orthogonal Laser Beams: A First Step Towards Internal/External Map Labeling**. In: *Canadian Conference on Computational Geometry (CCCG'10)*. 2010, 203–206.

[LNS15] Maarten Löffler, Martin Nöllenburg, and Frank Staals. **Mixed Map Labeling**. In: *Algorithms and Complexity (CIAC'15)*. Vol. 9079. Lecture Notes in Computer Science. Springer International Publishing, 2015, 339–351.

[LSC08] Martin Luboschik, Heidrun Schumann, and Hilko Cords. **Particle-Based Labeling: Fast Point-Feature Labeling without Obscuring other Visual Features**. *IEEE Transactions on Visualization and Computer Graphics* 14:6 (2008), 1237–1244.

[LWY09] Chun-Cheng Lin, Hsiang-Yun Wu, and Hsu-Chun Yen. **Boundary Labeling in Text Annotation**. In: *Information Visualization (InfoVis'09)*. IEEE Computer Society, 2009, 110–115.

[Mad+16] Jacob Boesen Madsen, Markus Tatzgern, Claus B. Madsen, Dieter Schmalstieg, and Denis Kalkofen. **Temporal Coherence Strategies for Augmented Reality Labeling**. *IEEE Transactions on Visualization and Computer Graphics* 22:4 (2016), 1415–1423.

[Mar05] Dániel Marx. **Efficient Approximation Schemes for Geometric Problems?** In: *Algorithms (ESA'05)*. Vol. 3669. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2005, 448–459.

[MD06] Stefan Maass and Jürgen Döllner. **Efficient View Management for Dynamic Annotation Placement in Virtual Landscapes**. In: *Smart Graphics (SG'06)*. Vol. 4073. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2006, 1–12.

[MD07] Stefan Maass and Jürgen Döllner. **Embedded Labels for Line Features in Interactive 3D Virtual Environments**. In: *Computer Graphics, Virtual Reality, Visualisation and Interaction (AFRIGRAPH'07)*. ACM Press, 2007, 53–59.

[Mil56] George A. Miller. **The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information**. *Psychological Review* 63:2 (1956), 81–97.

[Mog+12] Katja Mogalle, Christian Tietjen, Grzegorz Soza, and Bernhard Preim. **Constrained Labeling of 2D Slice Data for Reading Images in Radiology**. In: *Visual Computing for Biomedicine (VCBM'12)*. Eurographics Association, 2012, 131–138.

[Mor80] Joel L. Morrison. **Computer Technology and Cartographic Change**. In: *The Computer in Contemporary Cartography*. Johns Hopkins University Press, 1980, 5–24.

[Mot07] Kevin Mote. **Fast Point-Feature Label Placement for Dynamic Visualizations**. *Information Visualization* 6:4 (2007), 249–260.

[MP09] Konrad Mühler and Bernhard Preim. **Automatic Textual Annotation for Surgical Planning**. In: *Vision, Modeling, and Visualization (VMV'09)*. Eurographics Association, 2009, 277–284.

[MS91] Joe Marks and Stuart Shieber. **The Computational Complexity of Cartographic Label Placement**. Tech. rep. Harvard Computer Science Group, 1991.

[Nie12] Benjamin Niedermann. **Consistent Labeling of Dynamic Maps Using Smooth Trajectories**. Diploma Thesis. Karlsruhe Institute of Technology, 2012.

[NN16] Benjamin Niedermann and Martin Nöllenburg. **An Algorithmic Framework for Labeling Road Maps**. In: *Geographic Information Science (GIScience'16)*. Vol. 9927. Lecture Notes in Computer Science. Springer International Publishing, 2016, 308–322.

[NNR17] Benjamin Niedermann, Martin Nöllenburg, and Ignaz Rutter. **Radial Contour Labeling with Straight Leaders**. In: *Pacific Visualization Symposium (PacificVis'17)*. Accepted for publication. IEEE Computer Society, 2017.

[NPS10] Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. **Dynamic One-Sided Boundary Labeling**. In: *Advances in Geographic Information Systems (ACM-GIS'10)*. ACM Press, 2010, 310–319.

[NW00] Gabriele Neyer and Frank Wagner. **Labeling Downtown**. In: *Algorithms and Complexity (CIAC'00)*. Vol. 1767. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2000, 113–124.

[NW11] Martin Nöllenburg and Alexander Wolff. **Drawing and Labeling High-Quality Metro Maps by Mixed-Integer Programming**. *IEEE Transactions on Visualization and Computer Graphics* 17:5 (2011), 626–641.

[OP14] Steffen Oeltze-Jafra and Bernhard Preim. **Survey of Labeling Techniques in Medical Visualizations**. In: *Visual Computing for Biomedicine (VCBM'14)*. Eurographics Association, 2014, 199–208.

[Pet+09a] Stephen D. Peterson, Magnus Axholt, Matthew Cooper, and Stephen R. Ellis. **Evaluation of Alternative Label Placement Techniques in Dynamic Virtual Environments**. In: *Smart Graphics (SG'09)*. Vol. 5531. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2009, 43–55.

[Pet+09b] Stephen D. Peterson, Magnus Axholt, Matthew Cooper, and Stephen R. Ellis. **Visual Clutter Management in Augmented Reality: Effects of Three Label Separation Methods on Spatial Judgments**. In: *3D User Interfaces*. IEEE Computer Society, 2009, 111–118.

[PF96] I. Pinto and Herbert Freeman. **The Feedback Approach to Cartographic Areal Text Placement**. In: *Structural and Syntactical Pattern Recognition (SSPR'96)*. Vol. 1121. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1996, 341–350.

[PGP03] Ingo Petzold, Gerhard Gröger, and Lutz Plümer. **Fast Screen Map Labeling – Data Structures and Algorithms**. In: *International Cartographic Conference (ICC'03)*. 2003, 288–298.

[Poo+04] Sheung-Hung Poon, Chan-Su Shin, Tycho Strijk, Takeaki Uno, and Alexander Wolff. **Labeling Points with Weights**. *Algorithmica* 38:2 (2004), 341–362.

[PRS97] Bernhard Preim, Andreas Raab, and Thomas Strothotte. **Coherent Zooming of Illustrations with 3D Graphics and Text**. In: *Graphics Interface (GI'97)*. Canadian Human-Computer Communications Society, 1997, 105–113.

[Pul06] Wayne Pullan. **Phased Local Search for the Maximum Clique Problem**. *Journal of Combinatorial Optimization* 12:3 (2006), 303–323.

[Pul09] Wayne Pullan. **Optimisation of Unweighted/Weighted Maximum Independent Sets and Minimum Vertex Covers**. *Discrete Optimization* 6:2 (2009), 214–219.

[PW13] Friedrich Paulsen and Jens Waschke. **Sobotta Atlas of Human Anatomy, Vol. 3, 15th ed., English/Latin: Head, Neck and Neuroanatomy**. Elsevier Health Sciences Germany, 2013.

[QZ02] Zhongping Qin and Binhai Zhu. **A Factor-2 Approximation for Labeling Points with Maximum Sliding Labels**. In: *Algorithm Theory (SWAT'02)*. Vol. 2368. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2002, 100–109.

[RCS86] Raghunath Raghavan, James Cohoon, and Sartaj Sahni. **Single Bend Wiring**. *Journal of Algorithms* 7:2 (1986), 232–257.

[Roe89] Jan W. van Roessel. **An Algorithm for Locating Candidate Labeling Boxes Within a Polygon**. *The American Cartographer* 16:3 (1989), 201–209.

[RR16] Maxim Rylov and Andreas Reimer. **A Practical Algorithm for the External Annotation of Area Features**. *The Cartographic Journal* (2016). In press., 1–16.

[SB04] Monika Sester and Claus Brenner. **Continuous Generalization for Visualization on Small Mobile Devices**. In: *Spatial Data Handling (SDH'04)*. Springer Berlin/Heidelberg, 2004, 355–368.

[Sch+15] Nadine Schwartges, Benjamin Morgan, Jan-Henrik Haunert, and Alexander Wolff. **Labeling Streets Along a Route in Interactive 3D Maps Using Billboards**. In: *AGILE 2015*. Lecture Notes in Geoinformation and Cartography. Springer International Publishing, 2015, 269–287.

[SD08] Thierry Stein and Xavier Décoret. **Dynamic Label Placement for Improved Interactive Exploration**. In: *Non-Photorealistic Animation and Rendering (NPAR'08)*. ACM Press, 2008, 15–21.

[Sto+11] Jonathan Stott, Peter Rodgers, Juan Carlos Martinez-Ovando, and Stephen G. Walker. **Automatic Metro Map Layout Using Multicriteria Optimization**. *IEEE Transactions on Visualization and Computer Graphics* 17:1 (2011), 101–114.

[Str01] Tycho Strijk. **Geometric Algorithms for Cartographic Label Placement**. Dissertation. Utrecht University, 2001.

[SU02] Sebastian Seibert and Walter Unger. **The Hardness of Placing Street Names in a Manhattan Type Map**. *Theoretical Computer Science* 285 (2002), 89–99.

[SWH14] Nadine Schwartges, Alexander Wolff, and Jan-Henrik Haunert. **Labeling Streets in Interactive Maps using Embedded Labels**. In: *Advances in Geographic Information Systems (ACM-GIS'14)*. ACM Press, 2014, 517–520.

[Tat+14] Markus Tatzgern, Denis Kalkofen, Raphael Grasset, and Dieter Schmalstieg. **Hedgehog Labeling: View Management Techniques for External Labels in 3D Space**. In: *Virtual Reality (VR'14)*. IEEE Computer Society, 2014, 27–32.

[TKS13] Markus Tatzgern, Denis Kalkofen, and Dieter Schmalstieg. **Dynamic Compact Visualizations for Augmented Reality**. In: *Virtual Reality (VR'13)*. IEEE Computer Society, 2013, 3–6.

[Tuf01] Edward R. Tufte. **The Visual Display of Quantitative Information**. Graphics Press, 2001.

[Vol+07] Ian Vollick, Daniel Vogel, Maneesh Agrawala, and Aaron Hertzmann. **Specifying Label Layout Style by Example**. In: *User Interface Software and Technology (UIST'07)*. ACM Press, 2007, 221–230.

[VTW12] Mikael Vaaraniemi, Marc Treib, and Rüdiger Westermann. **Temporally Coherent Real-time Labeling of Dynamic Scenes**. In: *Computing Geospatial Research Applications (COM.Geo'12)*. ACM Press, 2012, 17:1–17:10.

[War+02] Colin Ware, Helen Purchase, Linda Colpoys, and Matthew McGill. **Cognitive Measurements of Graph Aesthetics**. *Information Visualization* 1:2 (2002), 103–110.

[War12] Colin Ware. **Information Visualization: Perception for Design**. 3rd. Morgan Kaufmann, 2012.

[WC11] Yu-Shuen Wang and Ming-Te Chi. **Focus+Context Metro Maps**. *IEEE Transactions on Visualization and Computer Graphics* 17:12 (2011), 2528–2535.

[Wol+00] Alexander Wolff, Lars Knipping, Marc van Kreveld, Tycho Strijk, and Pankaj K. Agarwal. **A Simple and Efficient Algorithm for High-Quality Line Labeling**. In: *Innovations in GIS VII: GeoComputation*. Taylor & Francis, 2000. Chapter 11, 147–159.

[Wol13] Alexander Wolff. **Graph Drawing and Cartography**. In: *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2013. Chapter 23, 697–736.

[WW97] Frank Wagner and Alexander Wolff. **A Practical Map Labeling Algorithm**. *Computational Geometry: Theory and Applications* 7:5 (1997), 387–404.

[YI13] Yusuke Yokosuka and Keiko Imai. **Polynomial Time Algorithms for Label Size Maximization on Rotating Maps**. In: *Canadian Conference on Computational Geometry (CCCG'13)*. 2013, 187–192.

[Yoe72] Pinhas Yoeli. **The Logic of Automated Map Lettering**. *The Cartographic Journal* 9:2 (1972), 99–108.

[Zha+15] Xiao Zhang, Sheung-Hung Poon, Minming Li, and Victor Lee. **On Maxmin Active Range Problem for Weighted Consistent Dynamic Map Labeling**. In: *GEOProcessing 2015*. IARIA, 2015, 32–37.

[Zor86] Steven Zoraster. **Integer Programming Applied to the Map Label Placement Problem**. *Cartographica* 23:3 (1986), 16–27.

List of Publications

Journal Articles

[1] **Multi-sided Boundary Labeling**. *Algorithmica* 76:1 (2016), 225–258. Joint work with Philipp Kindermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff.

Conference Articles

[2] **On the Readability of Boundary Labeling**. In: *Graph Drawing and Network Visualization (GD'15)*. Vol. 9411. Lecture Notes in Computer Science. Springer International Publishing, 2015, 515–527. Joint work with Lukas Barth, Andreas Gemsa, and Martin Nöllenburg.

[3] **Temporal Map Labeling: A New Unified Framework with Experiments**. In: *Advances in Geographic Information Systems (ACM-GIS'16)*. ACM Press, 2016. Joint work with Lukas Barth, Andreas Gemsa, and Martin Nöllenburg.

[4] **Using ILP/SAT to Determine Pathwidth, Visibility Representations, and other Grid-Based Graph Drawings**. In: *Graph Drawing (GD'13)*. Vol. 8242. Lecture Notes in Computer Science. Springer International Publishing, 2013, 460–471. Joint work with Therese Biedl, Thomas Bläsius, Martin Nöllenburg, Roman Prutkin, and Ignaz Rutter.

[5] **Towards a Topology-Shape-Metrics Framework for Ortho-Radial Drawings**. In: *European Workshop on Computational Geometry (EuroCG'17)*. Preprint. 2017. Joint work with Lukas Barth, Ignaz Rutter, and Wolf Matthias.

[6] **Towards a Topology-Shape-Metrics Framework for Ortho-Radial Drawings**. In: *Computational Geometry (SoCG'17)*. Leibniz International Proceedings in Informatics (LIPIcs). Accepted for publication. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. Joint work with Lukas Barth, Ignaz Rutter, and Matthias Wolf.

[7] **Trajectory-Based Dynamic Map Labeling**. In: *European Workshop on Computational Geometry (EuroCG'13)*. Preprint. 2013. Joint work with Andreas Gemsa and Martin Nöllenburg.

[8] **Trajectory-Based Dynamic Map Labeling**. In: *Algorithms and Computation (ISAAC'13)*. Vol. 8283. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 413–423. Joint work with Andreas Gemsa and Martin Nöllenburg.

[9] **Label Placement in Road Maps**. In: *European Workshop on Computational Geometry (EuroCG'14)*. Preprint. 2014. Joint work with Andreas Gemsa and Martin Nöllenburg.

[10] **Label Placement in Road Maps**. In: *Algorithms and Complexity (CIAC'15)*. Vol. 9079. Lecture Notes in Computer Science. Springer International Publishing, 2015, 221–234. Joint work with Andreas Gemsa and Martin Nöllenburg.

[11] **An Algorithmic Framework for Labeling Network Maps**. In: *Computing and Combinatorics (COCOON'15)*. Vol. 9198. Lecture Notes in Computer Science. Springer International Publishing, 2015, 689–700. Joint work with Jan-Henrik Haunert.

[12] **Two-Sided Boundary Labeling with Adjacent Sides**. In: *European Workshop on Computational Geometry (EuroCG'13)*. Preprint. 2013. Joint work with Philipp Kindermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff.

[13] **Two-Sided Boundary Labeling with Adjacent Sides**. In: *Algorithms and Data Structures (WADS'13)*. Vol. 8037. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 463–474. Joint work with Philipp Kindermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff.

[14] **An Algorithmic Framework for Labeling Road Maps**. In: *Geographic Information Science (GIScience'16)*. Vol. 9927. Lecture Notes in Computer Science. Springer International Publishing, 2016, 308–322. Joint work with Martin Nöllenburg.

[15] **Radial Contour Labeling with Straight Leaders**. In: *Pacific Visualization Symposium (PacificVis'17)*. Accepted for publication. IEEE Computer Society, 2017. Joint work with Martin Nöllenburg and Ignaz Rutter.

[16] **Radial Contour Labeling with Straight Leaders**. In: *European Workshop on Computational Geometry (EuroCG'17)*. Preprint. 2017. Joint work with Martin Nöllenburg and Ignaz Rutter.

Theses

[17] **Consistent Labeling of Dynamic Maps Using Smooth Trajectories**. Diploma Thesis. Karlsruhe Institute of Technology, June 2012.

Posters

[18] **PIGRA - A Tool for Pixelated Graph Representations.** In: *Graph Drawing (GD'14)*. Vol. 8871. Lecture Notes in Computer Science. Springer International Publishing, 2014, 513–514. Joint work with Thomas Bläsius, Fabian Klute, and Martin Nöllenburg.

[19] **Labeling Curves with Curved Labels**. In: *Abstracts of the Schematic Mapping Workshop 2014*. 2014. Joint work with Jan-Henrik Haunert, Herman Haverkort, Arlind Nocaj, Aidan Slingsby, and Jo Wood.