Alaa A. K. Ismaeel

Dynamic Hierarchical Graph Drawing

Karlsruhe Institute of Technology

Alaa A. K. Ismaeel

Dynamic Hierarchical Graph Drawing

Dynamic Hierarchical Graph Drawing

by

Alaa A. K. Ismaeel

Karlsruhe Institute of Technology

Dissertation, Karlsruher Instituts für Technologie (KIT),
Fakultät für Wirtschaftswissenschaften, 2012
Tag der mündlichen Prüfung: 21. Mai 2012
Referent: Prof. Dr. Hartmut Schmeck, Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB), Karlsruher Instituts für Technologie (KIT), Karlsruhe, Germany.
Korreferent: Prof. Dr. Jürgen Branke, Warwick Business School, The University of Warwick, Coventry, United Kingdom.

Dynamic Hierarchical Graph Drawing

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften

(Dr. rer. pol.)

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

genehmigte

DISSERTATION

von

M.Sc. Alaa Aly Khalaf Ismaeel

aus El-Minia, Ägypten

Tag der mündlichen Prüfung: 21. Mai 2012
Referent: Prof. Dr. Hartmut Schmeck
Korreferent: Prof. Dr. Jürgen Branke

2012 Karlsruhe

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

رَبَّنَا لَا تُؤَاخِذْنَآ إِن نَّسِينَآ أَوْ أَخْطَأْنَاۚ

Our Lord! Condemn us not if we forget, or miss the mark!

صَدَقَ اللهُ العَظِيمُ

سورة البقرة – آية ٢٨٦

Surah Al-Baqara – Verse 286

Full of pride, this thesis is dedicated to

the souls of the Egyptian January 2011 revolution martyrs,
the souls of my mother and my brothers Abdelmohsen and Mohammad,
my father and brothers Ashraf and Osama,
my great wife and life partner Mariam,
and my children Alzhraa, Abdelmohsen and Ashraqat.

Acknowledgements

First and foremost, my praise and deepest thanks are to God the Almighty for guiding me on the right path. Without HIS help and care, my efforts would have gone astray.

Looking back, I feel immense gratitude to many people around me who have supported me while I was working on this Ph.D. thesis. The past four and a quarter years have given me many exciting, challenging and truly valuable experiences.

This thesis is the result of a scholarship to carry out PhD in Computer Science at Karlsruhe Institute of Technology (KIT) in Germany. The scholarship was financially supported by the Egyptian Government (Egyptian Ministry of Higher Education) through the Egyptian Culture Bureau and Study Mission in Berlin. I would like to thank all of the administrative staff in Berlin and in the Culture Affairs & Mission Sector in Cairo.

I am sincerely thankful to my supervisor, Prof. Dr. Hartmut Schmeck. Despite his busy schedule in an ever-expanding research group plus his many responsibilities at the KIT, he frequently had some time for valuable and effective discussions. I am really grateful for you, Prof. Schmeck, for accepting me as one of your PhD students and also for getting involved as a member in your Efficient Algorithms research group at the Institute AIFB. Furthermore, for your encouragement, guidance and support that enabled me to develop an understanding of the thesis.

Similarly, I would also like to thank Prof. Dr. Jürgen Branke, from the University of Warwick, United Kingdom. He accepted, without hesitation, the request to serve as a second reviewer on the examination committee and also for his valuable comments. Before moving to the United Kingdom, Prof. Branke was working at the Institute AIFB, at the KIT. The preparation of the PhD proposal of this thesis was based on creative discussions with him. Moreover, I cannot forget how he helped me in my first few days in Karlsruhe. I am really very grateful for him. Furthermore, many thanks go to Prof. Dr. Karl-Heinz Waldmann and Prof. Dr. Ute Werner, both from the Karlsruhe Institute of Technology (KIT), who served as examiner and chairman, respectively, on the examination committee.

At the AIFB intensive and collaborative cooperation has been carried out with my colleague Dr. Pradyumn Shukla during the last two years of the thesis. I am greatly indebted to you for your honestly, help and kind support. Furthermore, I am grateful to all of my colleagues with whom I have worked during the last few years. The current and former members of the research group "Efficient Algorithms" provided a pleasant and productive working atmosphere. I would like to express my thanks to Dr. Ingo Pänke, Felix Vogel, Lukas König, Dr. Sanaz Mostaghim, Dr. Holger Prothmann, Dr. Urban Richter, Dr. Andreas Kamper,

Dr. Lei Liu, Dr. André Wiesner, Florian Allerding, Birger Becker, Frederic Toussaint, Nugroho Fredivianus, Christian Hirsch, Sabrina Merkel, Marc Mültin, Daniel Pathmaperuma, Friederike Pfeiffer, Micaela Wünsche, Fabian Rigoll, Fredy Rios, and our secretary Mrs. Ingeborg Götz. Special thanks also go to Felix, Sanaz, Lukas, and my office partner Nug.

In the Graph Drawing community, it is an honour for me to thank Christian Bachmaier and Andreas Gleißner (University of Passau, Germany) for our valuable discussions and the implementation of the Gravisto graph visualization toolkit. Furthermore, I am grateful to Prof. Perter Eades (University of Sydney, Australia), Prof. Roberto Tamassia (Brown University, USA), Prof. Ulrik Brandes (University of Konstanz, Germany) for our valuable discussions and the advice they gave during the 2010 and 2011 Graph Drawing conferences. Also, thanks are presented to Dr. Helen C. Purchase (University of Glasgow, Scotland), for her valuable comments.

Moreover, for the non-scientific side of my thesis, I offer my sincerest regards and blessings to all of those who supported me in any capacity during the completion of this thesis. I am grateful to my family and friends in Egypt for their constant moral support. Special thanks go to my father, my brothers Ashraf and Osama, father-in-law, mother-in-law, and brothers and sisters in law, who always believed in me, supported me in every respect and for their constant encouragement and endless praying for me throughout the last few years.

Last but not least, words fail to express my deepest thanks to my wife and life partner Mariam. I am so proud to dedicate this thesis to her, as she always enriches my life with her cheerful, spirited approach to life and countless and endless support. Really, she has been there for me every time I needed her even in stressful and chaotic times. Finally, I would like to show my gratitude to my children Alzhraa, Abdelmohsen, and Ashraqat for their innocent smiles, funny reactions and for making my Karlsruhe years so wonderful and unforgettable.

Karlsruhe, May 2012 *Alaa A. K. Ismaeel*

Abstract

Graph Drawing addresses the geometric representation of graphs. It is motivated by application fields in which it is crucial to visualize structural information as graphs. The majority of research into graph drawing has been in regard to efficient algorithms for working with static graphs in which the graph structure does not change. In *dynamic* graphs, changes in the graph structure and/or its drawing at use time of the graph are possible by adding, deleting, or modifying vertices and/or edges. Many graph drawing scenarios are dynamic since they involve a repeated updating of the graph, and consequently its drawing, after certain change is executed.

One of the main objectives in dynamic graph drawing is preserving the user's *mental map*, which is minimizes the effort the user has to spend in order to become re-familiarized with the drawing after some actions have been executed. The recommended approach for preserving the user's mental map is to minimize the changes between the new and current drawings. Effectively, minimizing the changes involves computing a measure of similarity between the two drawings. Collectively, these measures are known as *difference metrics*. Some considered difference metrics are the Euclidean distance, relative distance, edge orthogonality, vertex width, and vertex ordering.

This thesis is concerned with the problem of proposing and validating difference metrics of dynamic hierarchical graph drawings. The existing approaches in this direction focus on how to execute an action on a dynamic hierarchical graph, or on measuring the difference between drawings of a dynamic hierarchical graph using difference metrics. We introduce a new general framework on how to formulate difference metrics for hierarchical graphs. This framework is based on distinguishing between the vertices and edges that are inserted into or deleted from the graph and those that are still shared in both drawings. The general form of a difference metric is formulated by considering the topological and geometric characteristics of hierarchical graphs and have also been applied to existing metrics that are used with non-hierarchical graph drawings. The proposed formulation could be applied to any hierarchical graph of any size and could be easily extended to different graph types.

Measuring the similarity between two different drawings should not be done with respect to some computations of similarity metrics only since the human judgement based on the perception of what appears *aesthetic* into account. So, an experimental user study is carried out in order to validate the introduced difference metrics based on user evaluations.

Another problem for drawing hierarchical graphs aesthetically is minimizing crossings between edges. Two approaches are considered in solving this problem. The first is the *layer-by-layer sweep* and the second is the *global* approach. In the layer-by-layer sweep approach, the

problem is solved repeatedly by computing the order of the vertices in each layer according to their connectivity in the upper and lower layer, then the algorithm moves to the next two layers. In the global approach, the order of vertices in all layers is computed simultaneously. A new algorithm is introduced, named the efficient barycenter, which empirically produces better results than some recent global algorithms and traditional layer-by-layer algorithms, in both the number of crossings and the execution time.

Finally, a new method for generating random hierarchical graphs is introduced. This method is based on a precise counting of the maximum number of edges in a hierarchical graph. Two generators for proper and generally non-proper hierarchical graph are introduced. Both generators control all the parameters of a hierarchical graph, like the number of layers, the number of vertices in each layer, the minimum number of outgoing edges of a vertex, the edge density and the ratio of the long edges in the graph.

Contents

Figures List

Tables List

Algorithms List

1 Introduction

Un bon croquis vaut mieux qu'un long discours.[1]

This thesis is concerned with drawing dynamic hierarchical graphs. It focuses on defining and validating some difference metrics for measuring the geometric difference between two drawings of a dynamic hierarchical graph.

In this introductory chapter, we present the background to the thesis and reveal our motivation for this work in dynamic hierarchical graph drawing. We present a brief overview of the topics investigated and outline the main contributions arising from our study. Furthermore, this chapter provides some basic foundations from the area of graph drawing and some conceptual notations and conventions shared by the subsequent chapters of this thesis. More specific definitions are found in the respective chapters.

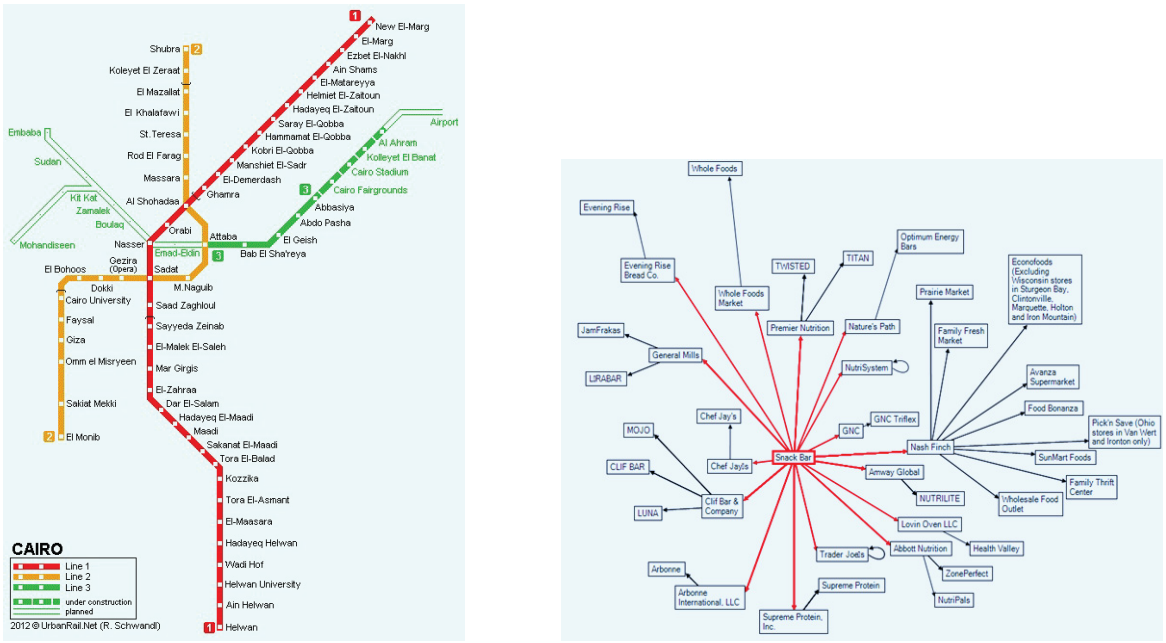
1.1 Graph Drawing

Graphs are discrete structures (see [GY99, Gro08]) used to represent information models that can be described as a set of objects and a set of relations or connections between those objects. In a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the objects are represented by the set of *vertices* $\mathcal{V}$ and the connections between those objects by the set of *edges* $\mathcal{E}$ that link the corresponding vertices.

[1] Napoleon Bonaparte. English translation: A good sketch is better than a long speech.

A *drawing* of a graph means a geometric representation of that graph in a plane by a 2-dimensional drawing or in space by a 3-dimensional drawing. In a *drawing* $\mathcal{D}$ of a graph $\mathcal{G}$, vertices represent states, components, agents, or modules and are drawn as *dots*, *boxes*, *circles* or other geometrical shapes where the edges are drawn using *arrows* or *lines* representing transitions, channels, or message transmissions. Arrows are used to show the direction of directed edges. The information corresponding to the vertices and edges can be visualized using text labels at various locations in or next to a graph object, may be in different colours, or other visual elements such as line thickness, box size, etc.

Many real-life applications consist of structures that can be modelled by graphs [BK02, CM08]. Some examples are biochemical reactions [SSL+09], email correspondences of organization members [SR04], maps visualization [GHK10, HGK10], electricity networks [GMS82, SPS10], international air traffic [DPS11], communication networks [Tri08], network visualization [WS98, MMB06], and the world wide web [Men96, HA99a], and the program evaluation and review technique PERT networks [Elm77, DPS11]. Two examples are presented in Figure 1.1. The drawing in Figure 1.1(a) shows the underground metro map of Cairo (the Egyptian capital) and the drawing represented in Figure 1.1(b) shows a network representing the connections between products and producers that may be tainted with peanuts that have food poisoning.



(a) (b)

Figure 1.1: Two examples of graph drawing in real life. Drawing (a) [Urb12] shows the Cairo underground metro map, where the vertices (stations) are represented by small white boxes and the edges (connections between stations) are represented as thick coloured straight-line segments. Drawing (b) [Smi09] represents the connections between products and producers that may be tainted with peanuts that have food poisoning, where the center vertex (Snack Bar) is the category pointing to the Recalling Firm which points to the Brand Name.

Graph drawing is an area of mathematics and computer science combining methods from geometric graph theory and information visualization to derive two-dimensional depictions of graphs arising from real-life applications. In other words, graph drawing addresses the problem of producing geometric representations of abstract graphs and networks and is motivated by those applications where it is crucial to visualize structural information as graphs. Since graph drawing methods form the algorithmic core of network visualization, bridging the gap between theoretical advances and implemented solutions is an important aspect.

The importance of graph drawing could be based on the Chinese proverb saying that *One picture is worth ten thousand words*, but this requires the picture to be clear and readable. Almost everybody is aware of graphs, composed of rectangles with information on them, and lines with arrows connecting them. Just think about the schematic representation of the organizational structure of a company. Or consider all relations and links in a database or a huge software program, which must be shown in a convenient way. Also, a plan for a project has to show clearly the underlying relationships between the tasks of the project, e.g., which tasks should be carried out concurrently or sequentially. Representing all of this information in a schematic diagram helps to manage the project.

A deeper and intensive information in the field of graph drawing have been introduced in several books, e.g., by Kamada [Kam89], Di Battista, Eades, Tamassia, and Tollis [DETT99], Kaufman and Wagner [KW01], Nishiziki and Rahman [NR04], and Tamassia [Tam07]. Most of the state-of-the-art graph drawing software tools are presented by Jünger and Mutzel in [JM04]. The latest research results can be found in the proceedings of the annual graph drawing conferences that appear in the *Lecture Notes in Computer Science* series of *Springer-Verlag* since 1992.

Graph drawing has been used for numerous and different real applications. Some examples include VLSI layout circuit designs [BC87, JG09], social networks [Sco00], bioinformatics [BGHM07, SH07], train and metro network maps [Wol07, NW11], subroutine-call graph visualization [GT02], software evolution visualization [CKN+03], entity-relationship database diagrams [Che76,BTT84,Auy90], data structures, computer security [TPP09], UML diagramming [BRJ99], and work flowchart management [Mur09].

The two Figures 1.2 and 1.3 (introduced in [JM04]) represent two drawings of the same graph. Figure 1.2 represents a newspaper drawing showing the network of the electric power industries concerning the dependencies of energy companies in Germany, where Figure 1.3 shows a more readable drawing, since it is without edge crossing, with shorter edge length, and with fewer edge bends.

The general graph drawing problem can be put simply:

> *Given a set of vertices $\mathcal{V}$ with a set of edges $\mathcal{E}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, calculate the position of every vertex $v \in \mathcal{V}$ and the curve representing each every $e \in \mathcal{E}$.*

Actually, this problem has always existed, for the simple reason that a graph is often defined by its drawing. Indeed, Euler (1707-1783) relied on a drawing to solve the *Königsberger Brücken problem* in his 1736 paper [Eul36]. The general problem of drawing graphs has extensively received a great deal of attention [Kam89,DETT94,DETT99,KW01,NR04,Tam07].

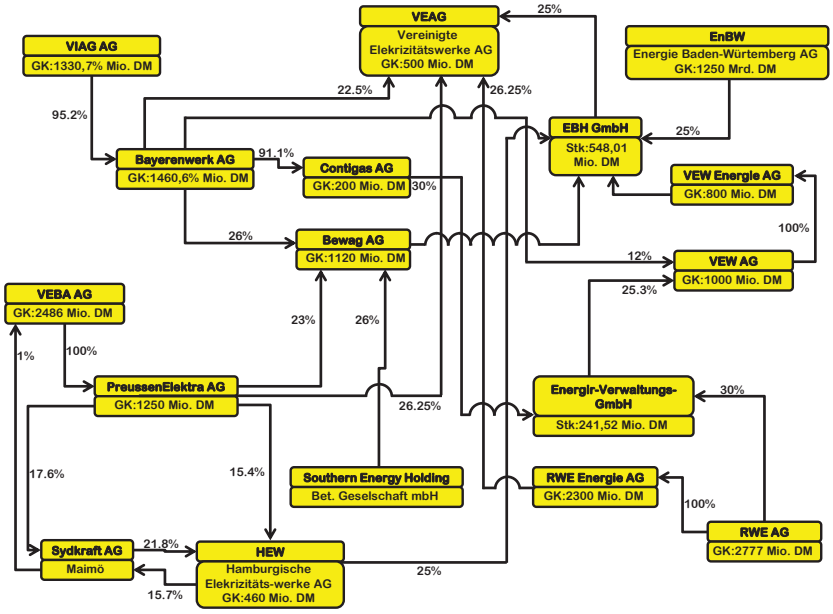


Figure 1.2: A newspaper diagram [JM04] representing the dependencies of energy companies in Germany, where the names of companies are written inside the yellow boxes and their interdependency percentage values are represented on the edges that connect the boxes.

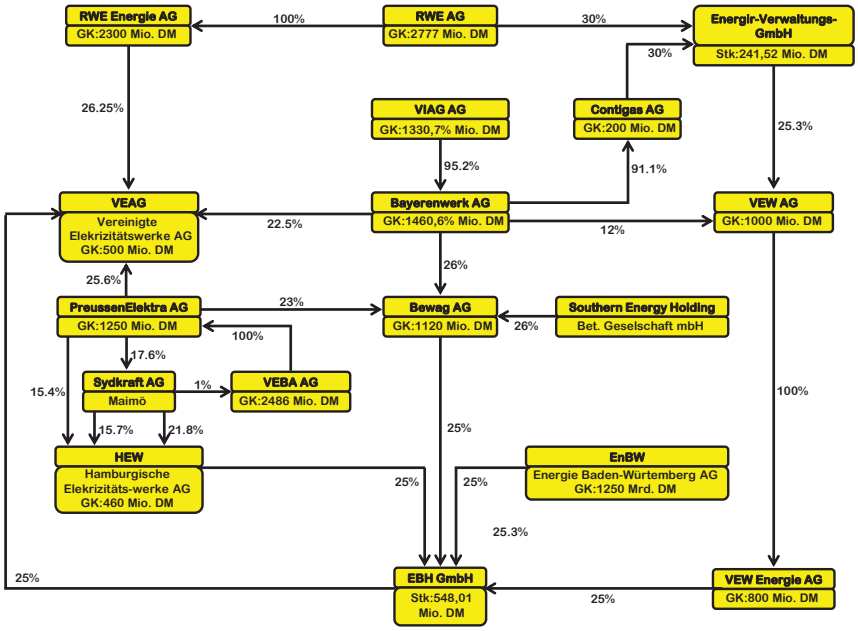


Figure 1.3: A better drawing [JM04] of the graph represented in Figure 1.2. The drawing here does not have edge crossings, has a shorter edge length, and also fewer edge bends.

The history of graph drawing is actually unknown, where the real start was in 1963 with the paper by Tutte titled "*How to draw a graph*" [Tut63], in which he suggested an algorithm that locates each vertex into the center of its neighbours. A recommended intuition says that a layout satisfying this drawing property is desirable, although it actually seems very difficult to model the *goodness* or *niceness* of a drawing. Nevertheless, there are some geometric pleasing factors (drawing criteria) that are widely understood as important for a *good* drawing. Some of these criteria are: small area, minimum edge crossings, maximum angles between edges, maximum symmetry, small number of edge bends, edge upwardness, graph clustering or aligning of related vertices etc.

The main goal of research in graph drawing is to develop techniques for constructing good drawings for the input abstract graphs. Informally speaking, a drawing is considered "*good*" if it is able to communicate effectively and clearly the information being displayed in the drawing. If a graph is small, then it can easily be drawn by hand, and the problem becomes more and more difficult if one tries to draw for example the diagram of an electrical network in a readable form. Here, the concept of *automatic graph drawing* arises. In automatic graph drawing, a computer program is using some graph algorithms in order to produce a final *readable* drawing of the graph. Automatic drawings are even more crucial for applications whose standard output is graphical, such as VLSI layouts, PERT networks, organization charts, etc.

The reason why automatic graph drawing is becoming more and more popular is in fact its broad application in different scientific areas, such as: biologists need to draw evolutionary trees, chemists need to draw large molecules, and architects need to draw their modelled designs. Also, databases are designed using entity relationship diagrams, and decision support systems for project management need to visualize PERT networks and activity trees. Last but not least, software engineers want data flow diagrams, subroutine-call graphs and object-oriented class hierarchies to be visualized.

1.2 Hierarchical Graphs

A fundamental issue in automatic graph drawing is to display hierarchical network structures as they appear in many applications such as social sciences [War76], graphical used interfaces [DETT94], VLSI layout [BL84], and many other applications. The network is transformed into a *directed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which has to be drawn with straight-line edges that are either all directed upwards or downwards. General directed acyclic graphs have been considered as not powerful enough to model every real-life application [ELT96, EFL97, Fen97, YS99] and hence *hierarchical graphs* are introduced. In hierarchical graphs, layering information is added to the directed acyclic graph that should have a hierarchical drawing.

In a hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$, a *layering* function ϕ is considered in order to partition the set of vertices $\mathcal{V}$ into a finite k subsets (*layers*) $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k$, such that the vertices belonging to the same layer $\mathcal{V}_i$ should be placed on the same horizontal line $L_i = i$, $1 \leq i \leq k$ and edges connect only vertices that belong to different layers, i.e., $\forall e = (u, v) \in \mathcal{E}, u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ this implies $i < j$. A more detailed introduction of hierarchical graphs will follow in Section 2.1.9. A hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$ could be

represented as $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k, \mathcal{E})$. Consequently, hierarchical drawing convention is proposed to display a specified layering information based on the direction of the edges. In order to get a useful hierarchical drawing, the drawing has to be a readable, understandable and easy to remember. Some of the aesthetic criteria of a good hierarchical drawing are: bends minimization, symmetry maximization, area minimization, and minimizing the number of edge crossings.

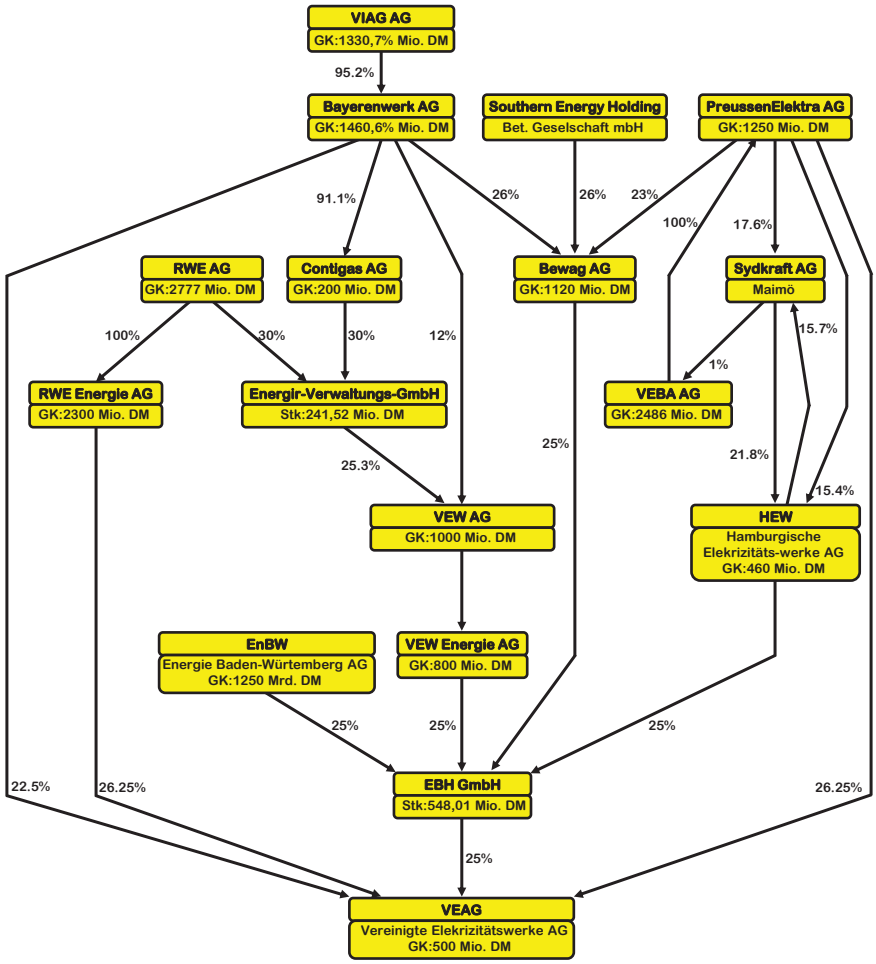
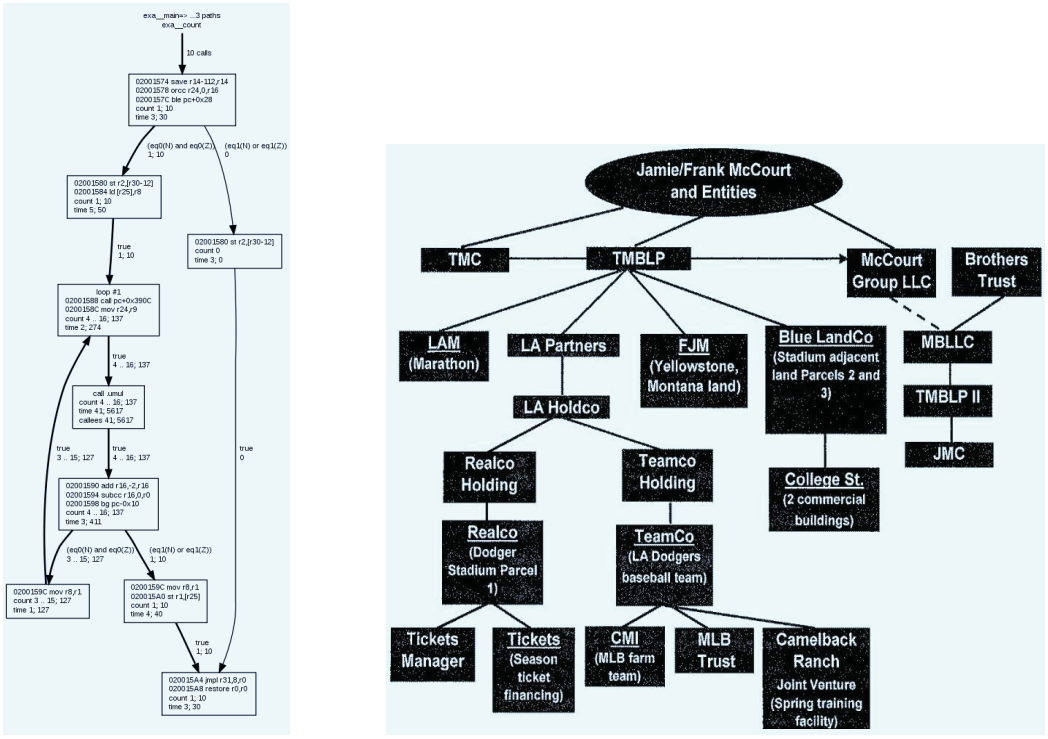


Figure 1.4: A traditional hierarchical drawing [JM04] of the graph represented in Figures 1.2 and 1.3 highlighting the dependency hierarchy of energy companies.

Drawing hierarchical graphs has many applications in many fields such as PERT networks [Elm77], subroutine-call graphs [Mye03], organizational charts [Gan03, Gan07], project management [CR06], visual languages [Cru93], interpretative structural modelling ISM [War76], entity-relationship database diagrams [Che76, BTT84, Auy90], software evolution diagrams [JK07, PVAE98], and data flow diagrams [BCD89]. The graph represented in Figure 1.4 represents a hierarchical drawing of the same graph of energy companies in Germany introduced in Figures 1.2 and 1.3. Furthermore, two examples of hierarchical drawings are represented in Figure 1.5. A flow graph of a program code is represented in Figure 1.5(a), where an organization chart for a company is shown in Figure 1.5(b).



(a) (b)

Figure 1.5: Two examples of hierarchical graph drawings in software engineering and relationship diagram: (a) represents a hierarchical drawing of a summary flow graph of exa___count total program code [Flo], and drawing (b) represents an organizational chart of the company McCourt Enterprise [Cha11].

A great deal of work has been done in producing hierarchical drawings of directed graphs [War77,Car80,STT81,Sug87,GM89,ELT96,San96,EFL97,Fri97,Lei98,Mut00,NW02,GBPD04, Ism04,EFN06,KPL06,Bac09,BBBH11,BBG11,DHW11,ZKS11,BBBF12]. Actually, the most common approach to draw directed graphs in a hierarchical manner was presented in 1981 by Sugiyama, Tagawa, and Toda [STT81] and called the *Sugiyama approach*. This approach consists of four main steps: *removing graph cycles by reversing some edges*, *assigning vertices to layers*, *reordering the vertices in each layer in order to minimize edge crossings*, and finally *assigning the horizontal coordinates for the vertices*. The Sugiyama approach and its four steps, with more focus on the third step of crossing minimization, is introduced in detail in Section 3.1.

An important criterion for many graph drawing applications is minimizing the number of edge crossings. Also, in many respects, crossing minimization is an exceptional problem in the wide range of optimization problems arising in automatic graph drawing generally and in drawing hierarchical graphs specifically. So, we concentrate on this problem and introduce it in Section 3.4. Furthermore, we introduce a new barycenter-based algorithm, name it the efficient barycenter, for minimizing crossings in hierarchical graphs. The proposed algorithm produces good results in both perspectives, the number of crossings and the running time.

1.3 Dynamic Graph Drawing

The majority of research in graph drawing has gone into efficient algorithms for working with *static* graphs. Static graphs are stable during the whole time span in which they are used, i.e. the graph and its structure remain unchangeable during the time span where the graph is used. An algorithm may be used in producing a drawing of the graph or just applying some computation. On the other hand, the problem becomes more difficult if the graph changes over time. In many scenarios, the considered graphs are *dynamic* and can continually change in their structure throughout the time span where they are used. In these dynamic scenarios, it is important to maintain coherence in the drawing, thereby helping the user to conserve his/her mental map [ELMS91] of the evolving information. The challenge here is to create a coherent sequence of drawings provide the necessary information.

When a user looks at a graph, he or she learns about the graph structure, and subsequently navigates through the drawing to understand its meaning. Hence, the user looking at drawings of a dynamic graph should be able to note changes being unfolded while maintaining an overall understanding of the data. The effort the user has to spend in order to become refamiliar with the new drawing , is termed as "*building the user's mental map*" and it is a main objective to minimize the user effort. However, piecing together a series of static snapshots is not sufficient in order to create drawings of a dynamic graph, since there is no consideration of preserving the mental map in this case.

Dynamic graph drawing provides many interesting research challenges. Starting from innovative ways of collecting data, moving to techniques of processing data to provide meaningful insights, and ending with creating cognition amplifying methods of displaying information. One's own social network is a good example of a dynamic network structure. The friends and acquaintances that one builds up throughout life form an interconnected mesh, since many of the people that you know will also know each other. Consequently and most importantly, the graph is constantly evolving as new people are encountered or old ties are severed. If data are available of the state of a social network at some different time steps, these snapshots can be linked together so that the evolution of the network can be seen. Another application area is the emerging field of pervasive and autonomic communication networks [DDF+06]. Pervasive systems, such as those in **smart rooms** which are aware of individuals and devices that enter and leave them, are typically concerned with ad-hoc topologies and transient communication channels. As devices enter the transmission range of the communication equipment in the system, they begin to transmit relevant contextual data, and in turn receive data from the system. These vertices are now changeable and will no longer be part of the network when the user moves out of range. A drawing of these interactions reveals an evolving network topology [SWQN07], which is useful for application designers to validate that contextual data is being disseminated correctly throughout their system.

The graph drawing problem gets worse if the graph changes over time by adding, deleting, or modifying vertices and/or edges. Here, the goal of the dynamic graph drawing is to preserve the user's mental map. Many researchers have addressed the dynamic graph drawing problem and preserving the mental map [MELS95,PST97,LLY06,PHG08,PS08,SP08,MR10,APP11a].

There are two ways used to preserve the user's mental map:

1. animating and highlighting the changes and hence the user can easily recognize them,
2. minimizing the changes between the two drawings in order to minimize the user effort needed to regain familiarity.

Although animation can be used to provide a smooth transition between consecutive drawings, it is still important to preserve some degree of similarity between drawings. If we cannot preserve the user's mental map, he or she may have to spend a lot of time relearning the new graph. So, the second way, minimizing the changes, is the more considered one in preserving the mental map. When inserting, deleting, or modifying some vertices and/or edges on a drawing of a graph, a new drawing is produced.

In order to minimize the changes between two drawings of a dynamic graph, the following two possibilities are considered:

1. Restricting the changes to the added or deleted vertices and edges keeping the remaining vertices and edges in both drawings the same,
2. Executing the changes freely on all the vertices and edges and then using some mathematical difference metrics to measure the change between the two drawings.

Restricting the changes to the added or deleted vertices and/or edges normally produces not so pleasing drawings since some aesthetic metrics could not be conveyed. For example, inserting a new edge may produce many crossings and then the readability of the drawing decreases. Instead, defining and validating difference metrics is the recommended approach in minimizing the changes.

Implementing experimental user studies offers a scientifically acceptable measure of a drawing performance. The major reason for using user studies is evaluating the strengths and weaknesses of different drawing techniques or drawing aesthetics. Studies can show whether a new visualization technique is useful in a practical sense, according to some objective criteria. User studies can objectively establish which aesthetic metric is more appropriate than other ones for a specific graph type. A more fundamental goal of conducting user studies is to seek insight into why a particular technique is effective. This can guide future efforts to improve existing techniques. We want to understand how aesthetic metrics and geometric properties yield high-quality results for a particular drawing. A good starting point in any study is the scientific or visual design question to be examined, and this drives the process of experimental design. Some experimental user studies have been carried out in order to validate the difference metrics forms [BT01, SP08, DLF+09, HvW09, PPP12].

For defining difference metrics, Bridgeman and Tamassia [BT00] introduced some formulations of difference metrics to measure the change between two orthogonal drawings of the same graph. Furthermore, Purchase [Pur02] presented formal metrics for quantifying the aesthetic presence in a graph drawing for seven common aesthetic criteria.

In this thesis, we study the problem of drawing hierarchical graphs that change in a dynamic scenario. In this main perspective, we study the problem of drawing dynamic hierarchical graphs including the following points:

- We study the way of executing an action on a hierarchical graph based on the hierarchical structure of the vertices and the layout of the edges curves, and the different possibilities of executing an action.
- We define a framework of a general difference metric form for measuring the difference between two drawings of a dynamic hierarchical graph based on the work presented in [BT00, Pur02].
- We conducted an experimental user study to validate the proposed difference metrics of dynamic hierarchical graph drawings.
- We consider the problem of minimizing edge crossings in drawing dynamic hierarchical graphs, i.e., after executing an action, the produced drawing should have the minimum number of crossings and at the same time should have minimum changes compared to the drawing prior to the action being executed.

1.4 Organization of the Thesis

This thesis consists of six chapters, of which this chapter is the first one. The remaining chapters are organized as follows:

Chapter 2 introduces some basic definitions of graph theory and graph drawing that we will deal with in this thesis. These basic definitions and terminology include graphs and its types and the most popular used drawing styles and techniques and their characteristics. Furthermore, some important drawing properties (criteria) and user requirements in the drawing.

In **Chapter 3**, we are addressing the problem of drawing hierarchical graphs, which is the graph type considered in this thesis. The Sugiyama approach, which is the most recommended method for drawing directed graphs hierarchically, is introduced. The four main steps of the Sugiyama approach are presented in details including some of the mainly considered techniques for each step. Also, we consider the problem of crossing minimization of hierarchical graphs and introduce a new advanced version of a barycenter-based algorithm, called the efficient barycenter algorithm, for this problem. The efficient barycenter algorithm outperforms the existing *layer-by-layer* techniques and also it produces results that are slightly better than the most recent technique called global sifting [BBBH11]. This chapter is closed by some notations about generating random hierarchical graphs.

Chapter 4 presents a main part of the work presented in this thesis, which is drawing dynamic hierarchical graphs. In this chapter, the basic concepts and terminology in the area dynamic graph drawing are introduced, including the definition of the user's mental map and its preservation methods. Some ideas about how to execute actions on hierarchical graphs in a dynamic environment are introduced, since these actions have different possibilities to be executed depending on the topological and geometric characteristics of hierarchical graphs. A general framework of distinguishing between the action and shared components (vertices and edges) of a hierarchical graph is proposed. Based on this general framework, a set of mathematical formulas for difference metrics is introduced in order to measure, more precisely, the geometric difference between two drawings of a dynamic hierarchical graph.

The proposed general framework could be applied to any graph type, rather than hierarchical graphs, and also with some modifications, the introduced difference metric notations could be applied to any other graph types. Also, a combination of the problem of minimize crossing and the dynamic drawing of hierarchical graphs is considered. This tries to answer the following question: Which drawing of a dynamic hierarchical graph has the maximum degree of similarity (or the minimum value of difference) to the original drawing and at the same time the minimum number of crossings?

Chapter 5 presents the results we got from an experimental user study, to validate the proposed difference metrics notations. That user study is performed in order to know how far the computed values for the mathematical forms of the proposed metrics are away from the user evaluation of the changes applied to hierarchical graphs in dynamic scenarios.

Finally, **Chapter 6** provided some concluding remarks and final comments on the work that is presented in this thesis and it contains some suggestions for potential directions of further investigations.

2 Basic Concepts

If there were only one truth, you couldn't paint a hundred canvases on the same theme.[1]

In this chapter we introduce some basic information and terminology in graph theory and graph drawing. The main intention of this chapter is to provide some notational conventions that we deal with in this thesis. The first section presents notations and mathematical definitions of graphs and its different models and types. The second section offers a brief description of the number of drawing styles and techniques that have been developed in the area of graph drawing. Finally, we introduce some properties of drawings and aesthetic criteria used in order to get a good, readable, or understandable drawing of a graph.

2.1 Graphs and Graph Types

2.1.1 Graphs

A *graph* $\mathcal{G}$ is a pair $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a finite set of *vertices* (also called *nodes* or *points*) and $\mathcal{E}$ is a finite set of *edges* (also called *links* or *connections*). Each edge $e \in \mathcal{E}$ consists of an unordered pair of distinct vertices $u, v \in \mathcal{V}$. The set $\mathcal{V}$ is called the *vertex set* of $\mathcal{G}$ and the set $\mathcal{E}$ is called the *edge set* of $\mathcal{G}$.

[1] Salvador Dali.

Let $e \in \mathcal{E}$ be an edge and $u \in \mathcal{V}$ be one of its two vertices, then e and u are called *incident* to each other. If $e \in \mathcal{E}$ is an edge with two incident vertices $u, v \in \mathcal{V}$, then edge e *connects* the vertices u and v, and we say that u and v are *adjacent vertices*. Two edges $e_1, e_2 \in \mathcal{E}$ that are incident to the same vertex $u \in \mathcal{V}$ are called *adjacent edges*. If $e \in \mathcal{E}$ is an edge incident to vertices $u, v \in \mathcal{V}$, we will use the notation $e = (u, v)$ to represent that edge e. An edge $e \in \mathcal{E}$ is called a *self-loop* (or *loop*) if it connects a vertex u to itself, i.e. $e = (u, u)$. Two edges $e_1, e_2 \in \mathcal{E}$ are called *parallel edges* (or *multiple edges*) if both connect the same two vertices, i.e. $e_1 = (u, v)$ and $e_2 = (u, v)$. A *simple graph* is a graph without *self-loops* or *parallel edges*, otherwise it is a *general graph* or simply a *graph*. Figure 2.1 represents a general graph with 5 vertices and 9 edges where the two edges b and c are parallel edges and edge i is a self-loop.

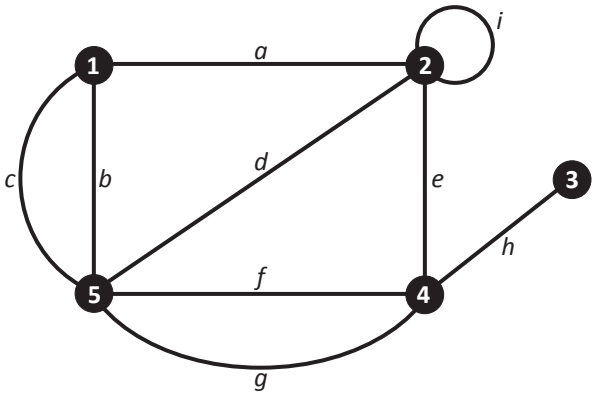


Figure 2.1: A graph with 5 vertices and 9 edges.

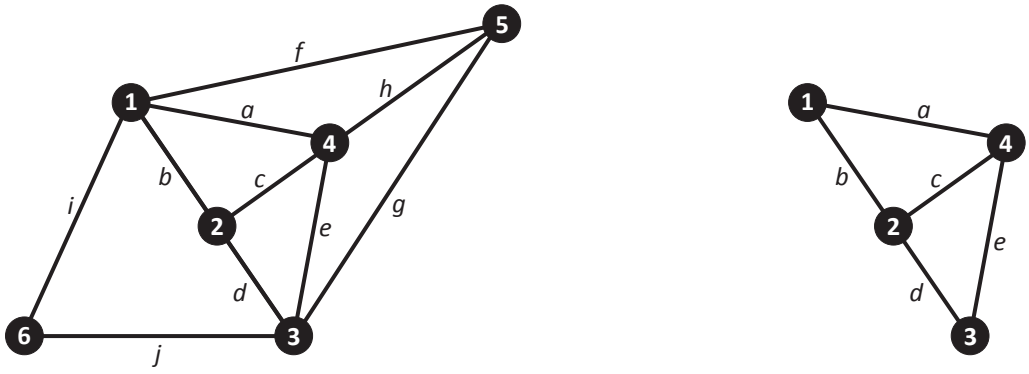
2.1.2 Directed Graphs

A *directed graph* (or *digraph*) $\mathcal{G}$ is a pair $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a finite set of vertices and $\mathcal{E}$ is a finite set of edges, where each edge $e \in \mathcal{E}$ consists of an ordered pair of vertices $u, v \in \mathcal{V}$. Ignoring for every edge the order of its vertices, we get an undirected graph that is called the *undirected graph* of $\mathcal{G}$. If $e = (u, v)$ is an edge in a directed graph $\mathcal{G}$, we say that e *leaves* vertex u and *enters* vertex v. Also, the vertex u is the *tail vertex* of $e = (u, v)$ and the vertex v is the *head vertex* of e, with e being the *outgoing edge* of u and the *incoming edge* of v. For an edge $e = (u, v)$, we say that u *dominates* v. A *source vertex* is a vertex with no incoming edges and a *target* (or *sink*) *vertex* is a vertex with no outgoing edges. A directed graph with one source vertex s and one target vertex t is called *st-digraph*.

The *indegree* $d^-(v)$ of a vertex v is the number edges entering v and the *outdegree* $d^+(v)$ of a vertex v is the number edges leaving v. In case of undirected graphs, the *degree* $d(v)$ of a vertex v is the number of its incident edges. A vertex v is called an *isolated vertex* if no edge is incident on it. Throughout this work, $n = |\mathcal{V}|$ denotes the number of vertices and $m = |\mathcal{E}|$ denotes the number of edges of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

2.1.3 Subgraphs

A graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is said to be a *subgraph* of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ or *contained* in $\mathcal{G}$ if $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$. If $\mathcal{V}' = \mathcal{V}$, $\mathcal{G}'$ is called a *spanning subgraph* of $\mathcal{G}$. If $\mathcal{E}'$ contains exactly those edges of $\mathcal{G}$ that connect two vertices in $\mathcal{V}'$ then $\mathcal{G}'$ is said to be *induced* by $\mathcal{V}'$. If $\mathcal{G}'$ is a subgraph of $\mathcal{G}$, then $\mathcal{G} - \mathcal{G}' = (\mathcal{V}, \mathcal{E} - \mathcal{E}')$ denotes the *difference* of $\mathcal{G}$ and $\mathcal{G}'$. Thus $\mathcal{G} - \mathcal{G}'$ is the subgraph of $\mathcal{G}$ induced by removing all edges that are in $\mathcal{G}'$. If $\mathcal{V} \subseteq \mathcal{V}'$, then $\mathcal{G} - \mathcal{V}'$ is the subgraph of $\mathcal{G}$ induced by $\mathcal{V} - \mathcal{V}'$. For a single vertex $v \in \mathcal{V}$, the graph $\mathcal{G} - \{v\}$ denotes the subgraph of $\mathcal{G}$ induced by $\mathcal{V} - \{v\}$. A directed graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is said to be a *directed subgraph* of a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ if $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$. If $\mathcal{V}' = \mathcal{V}$ then $\mathcal{G}'$ is called a *spanning directed subgraph* of $\mathcal{G}$. Figure 2.2(a) shows a graph of 6 vertices and 8 edges and one of its subgraphs is represented in 2.2(b).



(a) (b)

Figure 2.2: A graph (a) and one of its subgraphs (b).

2.1.4 Paths and Cycles

A *walk* w in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an alternating sequence of vertices and edges v_0, e_1, v_1, e_2, v_2, $\cdots e_k$, v_k begins and ends with two vertices $v_0, v_k \in \mathcal{V}$ and $e_i = (v_{i-1}, v_i) \in \mathcal{E}$, for $i = 1, 2, \cdots, k$. This walk w connects v_0 and v_k and may also be denoted by $w = (v_0, v_1, \cdots, v_k)$ with the edges being evident by context. A walk w is called a *path* if all vertices are distinct. The *length of a path* is the number of edges on that path. A walk is called a *cycle* (or *circle*) if all vertices are distinct except for $v_0 = v_k$ and $k \geq 3$. A graph without any cycles (as subgraphs) is called a *acyclic graph* (or *forest*).

A *directed walk* w in a directed graph $\mathcal{G}$ is a walk $w = (v_0, v_1, \cdots, v_k)$ in the underlying graph of $\mathcal{G}$ with $e_i = (v_{i-1}, v_i) \in \mathcal{E}$, for $i = 1, 2, \cdots, k$. An *undirected walk* w in a directed graph $\mathcal{G}$ is a walk in the underlying graph such that either $e_i = (v_{i-1}, v_i) \in \mathcal{E}$, or $e_i = (v_i, v_{i-1}) \in \mathcal{E}$ for $i = 1, 2, \cdots, k$. A directed or undirected walk w is called a *directed* or *undirected path* if all vertices are distinct. A directed walk w is called a *directed cycle* if all vertices are distinct except for $v_0 = v_k$. A directed graph that does not have any cycles is called a *directed acyclic graph* (*DAG*).

A directed acyclic graph with exactly one source vertex is called a *single source digraph*. Consequently, an acyclic digraph with exactly one target vertex is called a *single target digraph*. A directed acyclic graph with exactly one source s and exactly one target t is called an *st-dag* (or *st-directed acyclic graph*).

2.1.5 Graph Connectivity

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is *connected* if every pair of vertices is connected by a path. A *component* of $\mathcal{G}$ is a maximal connected subgraph of $\mathcal{G}$. Thus, a *disconnected graph* has at least two components. A *cut vertex* is a vertex whose removal increments the number of components. Thus, if $\mathcal{G}$ is connected, at least one vertex has to be removed from $\mathcal{G}$ in order to disconnect it. If no such cut vertex in $\mathcal{G}$ exists, $\mathcal{G}$ is called *biconnected* (or *2-connected*) graph. A pair of vertices $u, v \in \mathcal{V}$ is called a *split pair* if its removal disconnects the graph. The components that remain after the removal of a split pair u and v are called *split components* with respect to the vertices u and v. If no split pair in $\mathcal{G}$ exists, $\mathcal{G}$ is called *triconnected* (or *3-connected*) graph. A digraph is connected, biconnected or triconnected if its underlying graph is connected, biconnected or triconnected, respectively. Figure 2.3 shows a disconnected graph, which consists of two components.

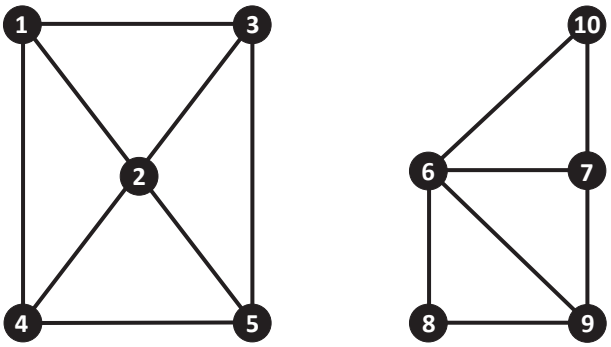
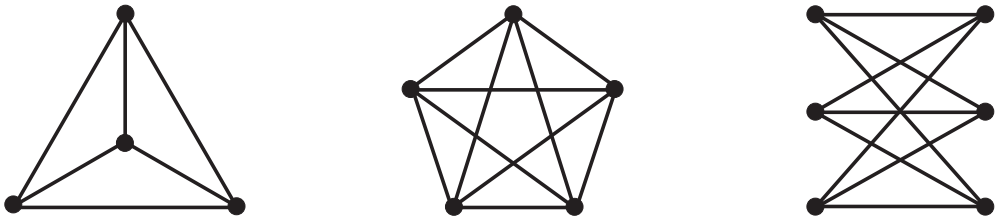


Figure 2.3: A disconnected graph with two components.

2.1.6 Complete and Bipartite Graphs

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is said to be a *complete graph* if every vertex $v \in \mathcal{V}$ is adjacent to every other vertex $w \in \mathcal{V} - \{v\}$. A complete graph with n vertices is denoted by K_n. Figure 2.4(a) shows the complete graph K_4, where Figure 2.4(b) shows the complete graph K_5. A *bipartite graph* $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ is a graph $(\mathcal{V}, \mathcal{E})$ whose vertex set $\mathcal{V}$ is partitioned into two sets $\mathcal{V}_1$ and $\mathcal{V}_2$ such that every edge connects a vertex in $\mathcal{V}_1$ and a vertex in $\mathcal{V}_2$. If every vertex in $\mathcal{V}_1$ is adjacent to every vertex in $\mathcal{V}_2$, $\mathcal{G}$ is a *complete bipartite graph*. Suppose $n_1 = |\mathcal{V}_1|$ and $n_2 = |\mathcal{V}_2|$, the complete bipartite graph is denoted by K_{n_1,n_2}. Figure 2.4(c) shows the complete bipartite graph $K_{3,3}$.



(a) The complete graph K_4 (b) The complete graph K_5 (c) The complete bipartite graph $K_{3,3}$

Figure 2.4: Example of complete and bipartite graphs.

2.1.7 Trees

A *tree* $\mathcal{T}$ is a connected graph without any cycles. Thus, a tree with n vertices has exactly $n-1$ edges. A *rooted tree* is a tree in which one of the vertices is distinguished from the others. The distinguished vertex is called the *root* of the tree. The root of a tree is generally drawn at the top. Figure 2.5 gives an example of a tree with 10 vertices where the root is vertex 1.

Every vertex u in a tree $\mathcal{T}$, other than the root vertex, is connected by an edge to some other vertex p called the *parent* of u and u is called a *child* of p. We draw the parent of a vertex above that vertex. For example, in Figure 2.5, vertex 1 is the parent of the two vertices 2 and 3, while vertex 2 is the parent of the two vertices 4 and 5. Also, the vertices 2 and 3 are children of vertex 1, while vertices 4 and 5 are children of 2. A *leaf* vertex is a vertex in a tree that has no children. An *internal vertex* is a vertex rather than the root vertex, that has one or more children. Thus, every vertex in a tree, rather than the root vertex, is either a leaf or an internal vertex. In Figure 2.5, the leaf vertices are 3, 5, 8, 9, and 10, and the internal vertices are 2, 4, 6 and 7. The *height* of a vertex u in a tree $\mathcal{T}$ is the length of a longest path from u to a leaf. So, height of a tree is the height of its root. The *depth* of a vertex u in a tree T is the length of a path from the root to u. In Figure 2.5, vertex 2 is of height 2 and depth 1, and the tree has height 3.

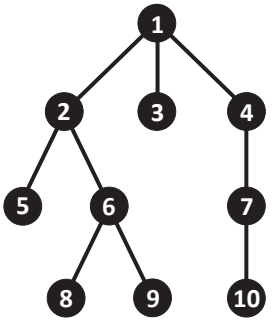


Figure 2.5: A tree with 9 vertices.

2.1.8 Planar Graphs

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is called *planar* if it can be drawn in the plane such that no two edges cross each other except at their common endpoints. A *planar embedding* of a planar graph $\mathcal{G}$ is an embedding with respect to a planar drawing. A graph with a given fixed planar embedding is also called a *plane graph*. Given any drawing with respect to a planar embedding of a graph $\mathcal{G}$, a *face* of $\mathcal{G}$ is any connected region in the drawing surrounded by the edges of $\mathcal{G}$. A face of a plane graph is uniquely described by its surrounding edges. The one unbounded face of a plane graph is called the *outer face* (or *exterior face*) and all other faces are called *inner faces* (or *interior faces*). The boundary of a face is the set of edges in the closure of the face.

In general, a planar graph has many planar embeddings in the plane, and two embeddings are said to be *equivalent* if the boundary of a face in one planar embedding always corresponds to the boundary of a face in the other planar embedding. A plane embedding of a graph is said to be *unique* if the planar embeddings are all equivalent.

2.1.9 Hierarchical Graphs

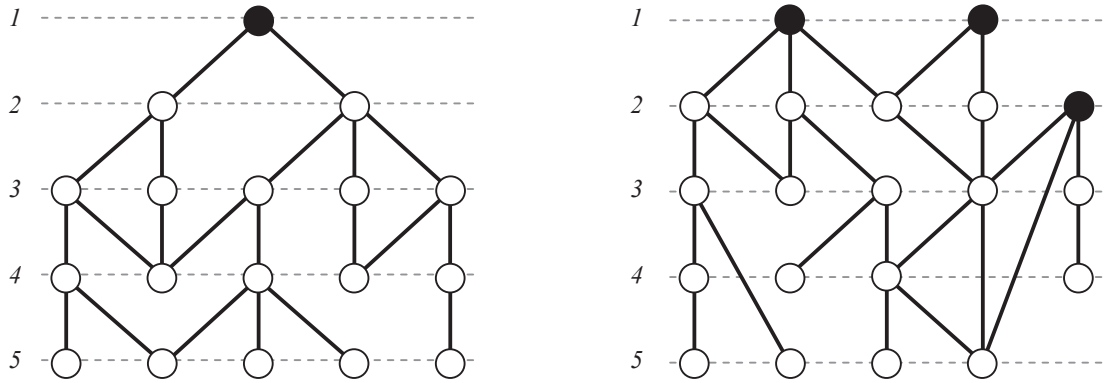
Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed acyclic graph. A *layering* (or *levelling*) of $\mathcal{G}$ is a topological numbering ϕ of $\mathcal{G}, \phi : \mathcal{V} \to \mathbb{Z}$, mapping the set of vertices $\mathcal{V}$ of $\mathcal{G}$ to integers such that $\phi(v) \geq \phi(u) + 1$ for every directed edge $(u, v) \in \mathcal{E}$ and $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$ is called a *hierarchical* (*layered* or *levelled*) *graph*. If $\phi(v) = j$, then v is a layer-j vertex and $\mathcal{V}_j = \phi^{-1}(j)$ is the j^{th} *layer* of $\mathcal{G}$. If $\mathcal{H}$ has a layering ϕ with k being the largest integer such that $\mathcal{V}_k$ is not empty, $\mathcal{H}$ is said to be a *k-layer hierarchical graph* and could be represented as $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$.

A hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$ is said to be *proper* if every edge $e \in \mathcal{E}$ connects only vertices belonging to consecutive layers. If a hierarchical graph is not proper, it must have at least one edge $e = (v, w) \in \mathcal{E}$ such that $v \in \mathcal{V}_i$ and $w \in \mathcal{V}_j$ with $1 \leq i < j - 1 \leq k - 1$, such an edge is called a *long edge* and it is said to be *traversing* the layers l with $i < l < j$. Any non-proper hierarchical graph can be transformed into a proper hierarchical graph by replacing every long edge with a path of short edges having a *dummy vertex* in every traversed layer. Figure 2.6 shows an example of proper and non-proper hierarchical graphs.

In a hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$, the two layers $\mathcal{V}_1$ and $\mathcal{V}_k$ are the *boundary layers* of $\mathcal{H}$. If all source vertices are in layer $\mathcal{V}_1$ and all sink vertices are in layer $\mathcal{V}_k$; in this case $\mathcal{H}$ is called a *boundary s-t hierarchical graph*. A mapping $\alpha : \mathcal{V} \to R^2$ is called a *sketch* and a *boundary sketch* β of $\mathcal{H}$ consists of an x-coordinate $\beta(v)$ for each vertex v in the boundary layers $\mathcal{V}_1$ and $\mathcal{V}_k$ of $\mathcal{H}$. A sketch α of $\mathcal{H}$ extends the boundary sketch β if $\alpha_x(v) = \beta_x(v)$ for each vertex $v \in \mathcal{V}_1 \cup \mathcal{V}_k$ in the boundary layers of $\mathcal{H}$.

2.1.10 Clustered Graphs

A *clustered graph* $\mathcal{C} = (\mathcal{G}, \mathcal{T})$ consists of an undirected graph $\mathcal{G}$ and a rooted tree $\mathcal{T}$ such that the leaves of the tree $\mathcal{T}$ are exactly the vertices of the undirected graph $\mathcal{G}$. Each vertex v of $\mathcal{T}$ (except the leaves) represents a cluster $\mathcal{C}(v)$ of the vertices of $\mathcal{G}$ that are leaves of the subtree rooted at v. Note that tree $\mathcal{T}$ describes an inclusion relation between clusters. The *height* of



(a) A proper hierarchical graph

(b) A non-proper hierarchical graph

Figure 2.6: Example of proper and non-proper hierarchical graphs. Source vertices are in black.

a cluster of vertex v of the tree $\mathcal{T}$, denoted $h(v)$, is defined as the depth of the subtree of $\mathcal{T}$ rooted at v. The *span* of an edge (u, v) of the tree $\mathcal{T}$ is $|h(u) - h(v)|$ where an edge of $\mathcal{T}$ of span greater than one is called a *long edge*.

In a plane drawing of a clustered graph $\mathcal{C} = (\mathcal{G}, \mathcal{T})$ (see Figure 2.7 [Fen97] as an example), graph $\mathcal{G}$ is drawn as points and curves in the plane as usual. For each vertex v of $\mathcal{T}$, the cluster is drawn as a simple closed region R that contains the drawing of $\mathcal{G}(v)$, such that:

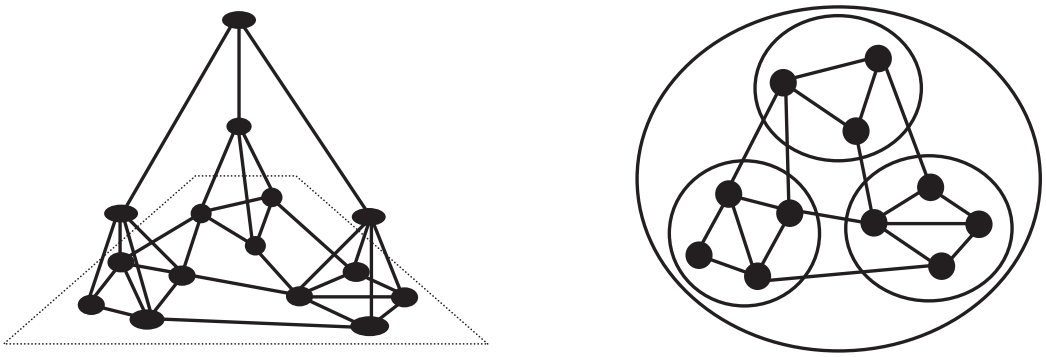
- The regions for all sub-clusters of R are completely contained in the interior of R.
- The regions of all other clusters are completely contained in the exterior of R.
- If there is an edge e between two vertices of $\mathcal{V}(v)$, then the drawing of e is completely contained in R.

2.2 Drawing Styles

A drawing $\mathcal{D}$ of a graph $\mathcal{G}$ is a geometric representation of the graph in the plane where vertices are represented by symbols such as shapes, circles, or points and edges are represented by simple curves such as line segment connecting the symbols that represent the associated vertices. In this section we introduce some important well-known drawing styles and related terminology.

2.2.1 Planar Drawing

A drawing of a graph is *planar* if no two edges intersect in the drawing except at their common end points. It is preferable to find a planar drawing of a graph if possible where unfortunately not all graphs admit planar drawings.



(a) A clustered graph in 3 dimensions

(b) 2-dimensional representation of the cluster graph in drawing (a)

Figure 2.7: Example of a cluster graph.

To find a planar drawing of a given graph, one needs to test whether the given graph is planar or not; this is called *graph planarity testing*. If the graph is planar, then finding a planar representation of the graph is needed. Hopcroft and Tarjan [HT74], and Booth and Lueker [BL76] developed linear-time algorithms for the graph planarity testing problem. Chiba, Nishizeki, Abe and Ozawa [CNAO85], Cai, Han and Tarjan [CHT89] and Mutzel [Mut92] gave linear-time algorithms for finding a planar representation of a planar graph. A planar graph with a fixed planar drawing is called a *plane graph*. Recent work in planar drawings is presented in [Ead08, GMZ09, Bie11, CG12]. Figure 2.8 shows a planar drawing in 2.8(a) and a non-planar drawing 2.8(b) of the same graph.

(a) (b)

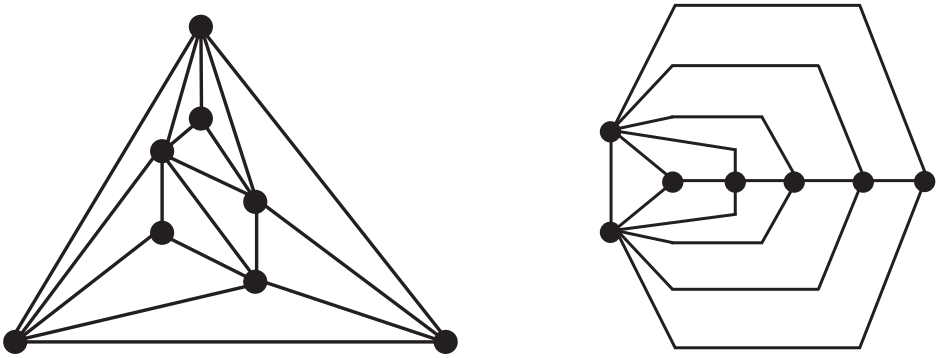
Figure 2.8: A planar drawing (a) and a non-planar drawing (b) of the same graph.

2.2.2 Straight-Line Drawing

A *straight-line drawing* is a drawing of a graph in which all edges of the graph are drawn as straight-line segments (see Figure 2.9(a)). Wagner [Wag36], Fáry [Far48] and Stein [Ste51] independently proved that every planar graph has a straight-line representation. Several researchers intensively investigated the problem of drawing planar graph with straight-line edges [DETT94, GR07, Kar09, BDD+10].

2.2.3 Polyline Drawing

A *polyline drawing* [GM98, ZS08, AFT11] is a drawing of a graph in which each edge of the graph is represented by a polygonal chain. A polyline drawing of a graph is shown in Figure 2.9(b). The point at which an edge changes its direction in a polyline drawing is called an *edge bend*. Polyline drawings provide great flexibility since they can approximate drawings with curved edges. However, edges with more than two or three bends may be difficult to follow for the eye. Note that a straight-line drawing is a special case of a polyline drawing, where edges are drawn without bends.



(a) (b)

Figure 2.9: A straight-line drawing (a) and a polyline drawing (b).

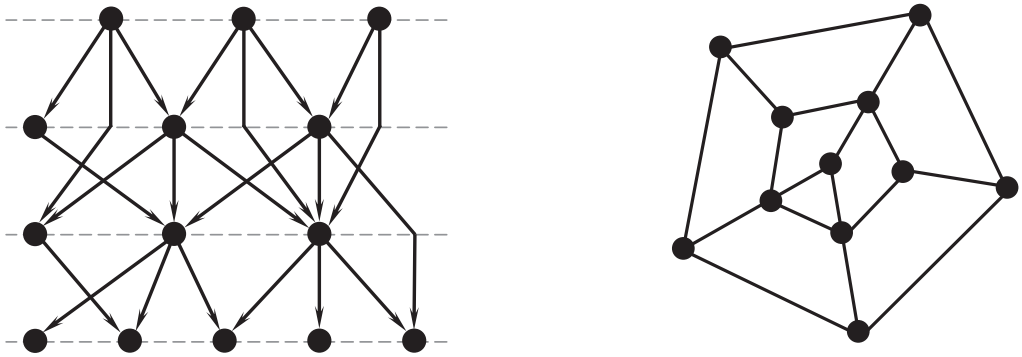
2.2.4 Hierarchical Drawing

A hierarchical drawing of a directed acyclic digraph is a downward (or upward) straight-line (or polyline) drawing where the vertices and bends are constrained to lie on a set of horizontal lines, called layers (see Figure 2.10(a)). In some applications the assignment of vertices to layers is given, e.g., by the semantics of the graph. Such graphs are called hierarchical (or layered) graphs. Edges in hierarchical drawing connect vertices belonging to different layers only. An edge is a *short edge* if it connects vertices belonging to consecutive layers, otherwise it is a *long edge*. Each long edge is broken into a chain of short edges by using a dummy vertex on each of the intermediate layers that are passed. Dummy vertices have all the attributes of normal vertices except that dummy vertices are represented as bends.

In a *hierarchical drawing* of a directed acyclic graph $\mathcal{G}$ in the plane the vertices of every layer $\mathcal{V}_j, 1 \leq j \leq k$, are placed on a horizontal line $L_j = \{(x, k-j) \mid x \in R\}$, and every edge $(u, v) \in \mathcal{E}$, $u \in \mathcal{V}_i$, $v \in \mathcal{V}_{i+1}$, $1 \leq i < j \leq k$, is drawn as a monotonic decreasing curve between the lines L_i and L_j. A hierarchical drawing of $\mathcal{G}$ is called *hierarchical planar* if no two edges cross except at common endpoints. A hierarchical graph is a *hierarchical planar graph* if it has a hierarchical planar drawing. A hierarchical graph obviously is hierarchical planar if and only if all its components are hierarchical planar. A graph that is not hierarchical planar is usually called a *non-hierarchical planar graph*. The general framework of drawing graphs hierarchically is presented in detail in Section 3.1.

2.2.5 Convex Drawing

A *convex drawing* is another way for drawing planar graphs by drawing it with convex polygons, i.e., a planar straight-line drawing such that all internal face boundaries are convex polygons. Figure 2.10(b) shows a convex drawing. Although not every graph has a convex drawing, every 3-connected plane graph has such a drawing [Tut60]. Several algorithms are known for finding a convex drawing of a plane graph [CYN84,Kan93,CGT96,RB06,BMNR10,Rot12].



(a) (b)

Figure 2.10: A hierarchical drawing (a) and a convex drawing (b).

2.2.6 Orthogonal Drawing

An *orthogonal drawing* is a drawing of a plane graph in which each edge is drawn as a chain of horizontal and vertical line segments (see Figure 2.11(a).) Orthogonal drawings have attracted much attention due to their numerous applications in circuit layouts, database diagrams, entity relationship diagrams etc. Many results have been published in recent years on orthogonal drawings [Sto84,Tam87,TTV91,BK94,PT95,Bie96,PT97,Bie98,ESW00, BDLN05,BKRW11,DKL$^+$12,KT12]. Clearly, each vertex in an orthogonal drawing is drawn as a point. Obviously a graph having a vertex of degree 5 or more has no orthogonal drawing, because at most four edges can be incident to a vertex in an orthogonal drawing.

A *box-orthogonal drawing* of a graph is a drawing such that each vertex is drawn as a possibly degenerate rectangle, called a *box*, and each edge is drawn as a sequence of alternate horizontal and vertical line segments, as illustrated in Figure 2.11(b). Every plane graph has a box-orthogonal drawing.

2.2.7 Rectangular Drawing

An orthogonal drawing is called a *rectangular drawing* if it has no bends and each face is drawn as a rectangle. Figure 2.11(c) shows a rectangular drawing of a graph. A rectangular drawing of a plane graph $\mathcal{G}$ is a drawing of $\mathcal{G}$ in which each vertex is drawn as a point, each edge is drawn as a horizontal or vertical line segment without edge crossings, and each

face is drawn as a rectangle. Not every plane graph has a rectangular drawing. Thomassen [Tho84], and Rahman, Nakano and Nishizeki [RNN02] established necessary and sufficient conditions for a plane graph of the maximum degree three to have a rectangular drawing. Linear-time algorithms for finding rectangular drawings of such plane graphs are also known [BS88, RNN98, RNN02]. Recently Miura, Haga and Nishizeki [MHN05, Nis07] reduced the problem of finding a rectangular drawing of a plane graph of the maximum degree four to a perfect-matching problem. A planar graph $\mathcal{G}$ is said to have a rectangular drawing if at least one of the plane embeddings of $\mathcal{G}$ has a rectangular drawing.

A *box-rectangular drawing* of a plane graph $\mathcal{G}$ is a drawing of $\mathcal{G}$ on the plane such that each vertex is drawn as a (possibly degenerate) rectangle, called a box, and the contour of each face is drawn as a rectangle, as illustrated in Figure 2.11(d). If $\mathcal{G}$ has multiple edges or a vertex of degree five or more, then $\mathcal{G}$ has no rectangular drawing but may have a box-rectangular drawing. However, not every plane graph has a box-rectangular drawing.

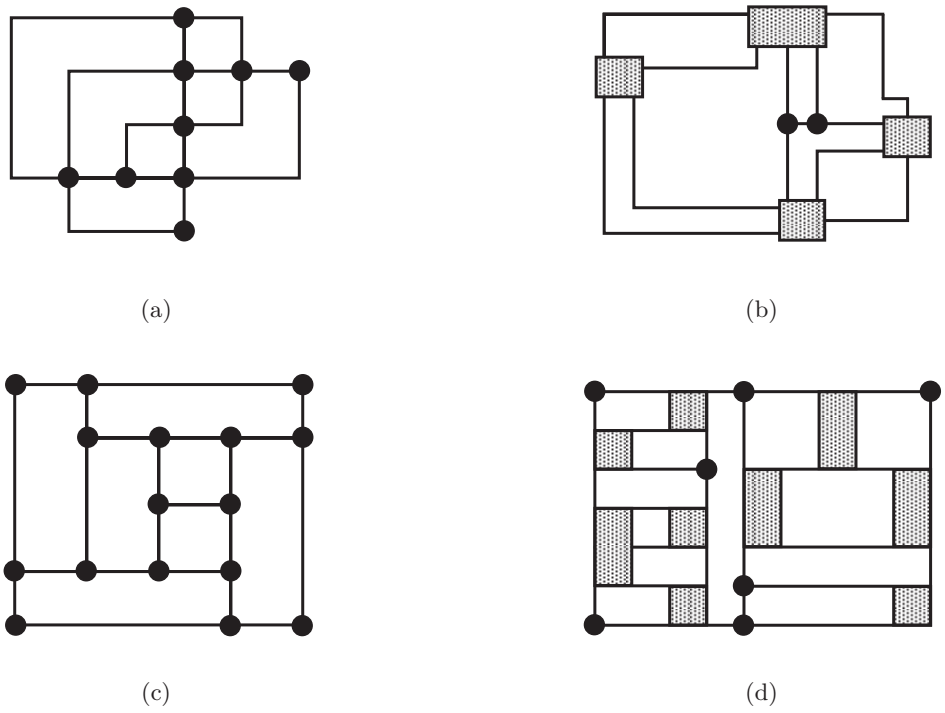


Figure 2.11: An orthogonal drawing (a), a box-orthogonal drawing (b), a rectangular drawing (c), and a box-rectangular drawing (d).

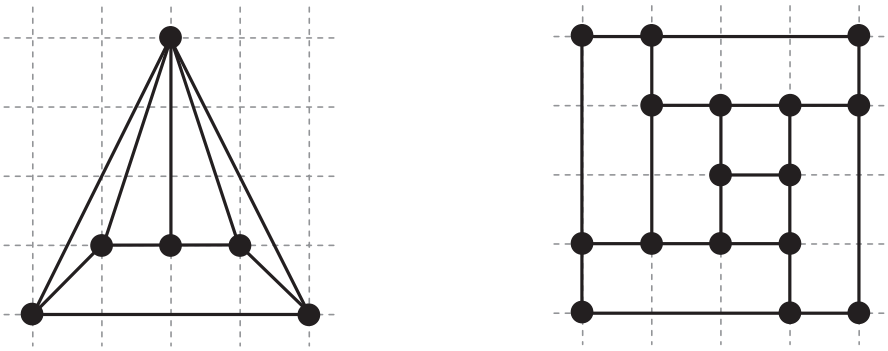
2.2.8 Grid Drawing

A drawing of a graph in which vertices and bends are located at grid points of an integer grid is called a grid drawing (as illustrated in Figure 2.12.) The grid drawing approach overcomes the following problems in graph drawing with real number arithmetic [Rah99]:

- When the embedding has to be drawn on a raster device, real vertex coordinates have to be mapped to integer grid points, and there is no guarantee that a correct embedding will be obtained after rounding.

- Many vertices may be concentrated in a small region of the drawing. Thus the embedding may be messy, and line intersections may not be detected.
- One cannot compare area requirement for two or more different drawings using real number arithmetic, since any drawing can be fitted into any area using magnification.

The size of an integer grid required for a grid drawing is measured by the size of the smallest rectangle on the grid that encloses the drawing. The width w of the grid is the width of the rectangle and the height h of the grid is the height of the rectangle. The grid size is usually described as $w \times h$. Drawing a plane graph on a grid of the minimum size is a very challenging problem, since it is a recommended drawing criterion in many real applications such as VLSI designs. Several results have been devoted to this direction [Tam87, FPP90, Sch90, BW98, CN98, RNN98, BCMW04, Kar09].



(a) (b)

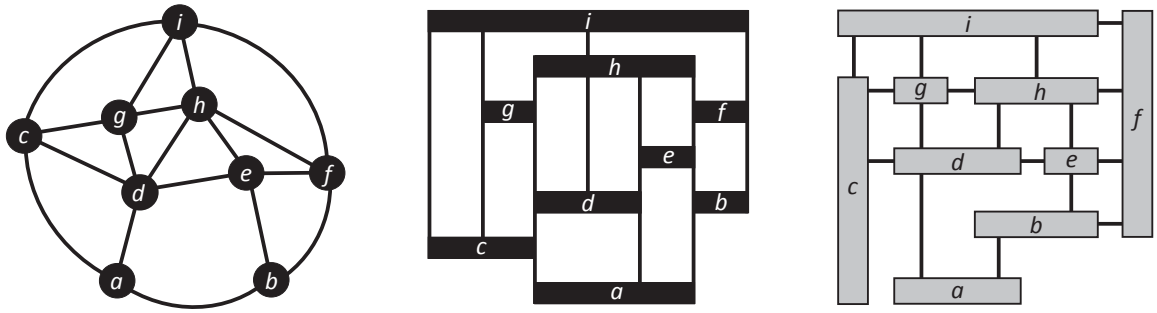
Figure 2.12: A straight-line grid drawing (a) and a rectangular grid drawing (b).

2.2.9 Visibility Drawing

In a *visibility drawing* (or *visibility representation*) of a planar graph vertices are represented as horizontal segments and edges as vertical segments such that each edge segment has its end points on the segments associated with its incident vertices and does not cross any other vertex segment. Otten and van Wijk [OW78] introduced this representation, which has applications to circuit schematics, and showed that every planar graph admits one. Note that instead of using horizontal segments for vertices representations, we can use rectangle or triangle segments. Some important results about visibility drawing may be found in [TT86,KLTT97,LE99]. In Figure 2.13, a graph and its visibility representations are presented.

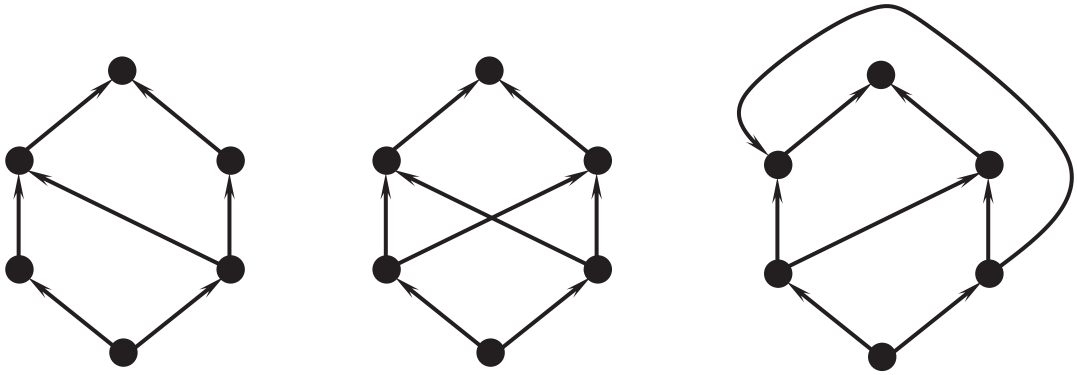
2.2.10 Upward Drawing

An upward drawing of a directed acyclic graph $\mathcal{G}$ is a drawing of $\mathcal{G}$ such that each edge is drawn as a curve monotonically increasing in the vertical direction. A digraph is upward planar if it admits a planar upward drawing (see Figure 2.14(a)). Figure 2.14(b) shows an upward drawing, which is not planar and Figure 2.14(c) shows a planar drawing which is not upward.



(a) (b) (c)

Figure 2.13: A plane drawing of a graph $\mathcal{G}$ (a), a visibility drawing $\mathcal{G}$ (b), and 2-visibility drawing of $\mathcal{G}$ (c).



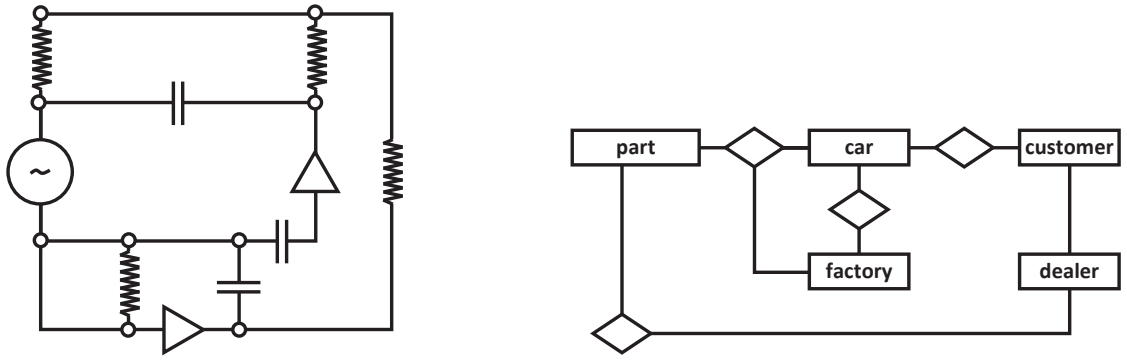
(a) (b) (c)

Figure 2.14: An example of upward drawings: an upward planar drawing (a), an upward non-planar drawing (b), and a non-upward planar drawing (c).

2.3 Drawing Aesthetic Criteria

There are infinitely many drawings for a graph. In drawing a graph, we would like to take into account a variety of properties. For example, we may be interested in a planar orthogonal drawing of a graph corresponding to a VLSI circuit such that the number of bends in the drawing is as small as possible, because bends increase the manufacturing cost of a VLSI chip, see [Wid83, BC87, Nis07, IRW99, JG09]. To avoid wasting valuable space in the chip, it is important to keep the area of the drawing small. Even when we are motivated to obtain only a nice drawing, we cannot precisely define what a nice drawing is, and hence we consider some properties of the drawing.

The most successful approach to developing algorithms satisfying the readability requirements uses the idea of *drawing aesthetics*. An attempt is made to identify those properties (aesthetics) which make a particular drawing style successful in representing specific class of graphs in an easily understood way. See the two drawings presented in Figure 2.15 [DETT99]. A simple example of a drawing aesthetic might be "edges should be drawn as straight lines of the same length". A drawing is accepted if it satisfies, or maximizes compliance with the chosen aesthetics.



(a) (b)

Figure 2.15: Example of planar orthogonal drawings: an electrical circuit schematic (a) and an entity-relationship diagram (b).

Aesthetics for graph layout can be divided into three categories [EL93]:

1. **Global criteria:** such as minimizing the number of edge crossings, maximizing symmetry, or averaging the graph edge length.
2. **Correctness criteria:** such as placing the employer above the employees in an organization tree drawing, or vertex v should be on the right of vertex u.
3. **Preferred criteria:** which express the preferences of a specific user at a specific time. These criteria may include placing a particular vertex in the centre of the page, or using a particular aspect ratio.

The choice of a set of drawing aesthetics is crucial. If the constraints imposed by the selected aesthetics are too loose, there may be no unique drawing, while if they are too severe, the situation may arise that no drawing satisfies all the aesthetics simultaneously. In the former case, additional constraints are needed to uniquely determine a drawing. In the latter case, the aesthetics conflict and a compromise between them is required, so that the drawing approximately satisfies all the aesthetics. An optimum choice (in this sense) prescribes a set of aesthetics, and assigns a relative importance to each, so that a unique drawing results [Ost96]. Unfortunately, there is no clear ranking among these criteria which would be valid for all possible applications.

In the following, we introduce some properties of graph drawings with a list of some commonly used aesthetics criteria together with examples of their practical importance. Note that these criteria depend entirely or mostly on the structure of the graph, so algorithms for

optimizing these criteria can be devised easily and plugged in as extension to improve the output of a graph drawing system. Commonly adopted aesthetics (see for example, the studies in [STT81, BFN85, PCJ96, Pur00, BT01, Pur02, SP08, DLF+09, HvW09, PPP12]) include the following items:

- **Readability**

  The information contained in the graph should be easy to read, i.e. it should effectively display the information of interest to the user. Most drawing techniques have approached this property by simultaneously enforcing *readability* criteria such as:

  - The drawing should exhibit structural properties of the graph, which are of particular interest to the user (for example, symmetries in the graph).
  - The drawing should maximize the angle between adjacent or crossing edges.
  - Vertices should be distributed uniformly in the drawing area with adequate vertex resolution, defined as the minimum distance between vertices in the drawing. Pairs of adjacent vertices should have similar separations.

- **Efficiency**

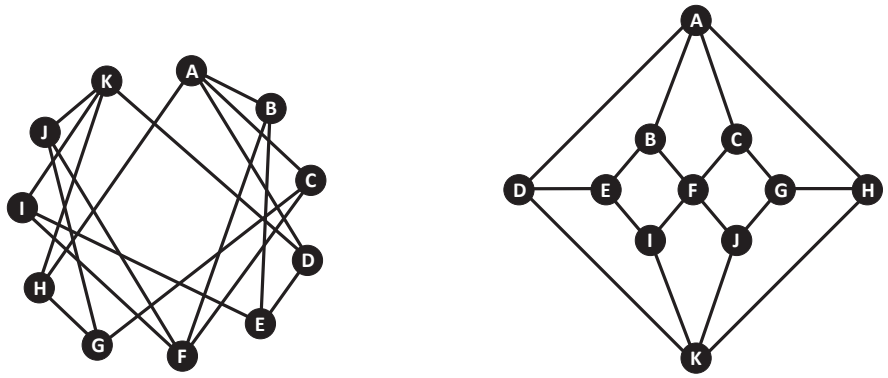
  Computational efficiency is an important property of any graph drawing algorithm. Interactive applications require real-time response, even for large drawings. Hence efficiency is a crucial issue for any practical graph drawing technique. The drawing algorithms should complete the display in a time acceptable to the user, typically within two seconds for an interactive use [Shn86].

- **Uniqueness**

  A graph should be drawn in a *unique* representation or invariant to its structure. In other words, a drawing of a digraph is independent of other conditions than its structure. This is a recommended convention in graph drawing since this helps the user to deal directly with the unique drawing instead of spending time in comparing multiple drawings of the same graph to select one of them.

- **Symmetry maximization**

  If a graph contains symmetrical information then it is important to reflect this symmetry in its layout. Technical drawings often contain hidden symmetries. Unfortunately, displaying symmetries is not an easy task. *Symmetry* is an important aesthetic criterion in graph drawing. A symmetry of a two-dimensional drawing is an isometry of the plane that fixes the drawing [HE03]. There are two types of two-dimensional symmetry, *rotational symmetry* and *reflectional symmetry*. Rotational symmetry is a rotation about a point where reflectional symmetry is a reflection about an axis. This aesthetic can be further formalized by introducing a mathematical model of symmetries in graphs and drawings (see, e.g., [LNS85, Ead88, PKL04, HL06, BL10]). For example, Figure 2.16 represents two different drawings of the same graph. The second drawing in Figure 2.16(b) gives a symmetric drawing of the considered graph, whereas the drawing in Figure 2.16(a) does not.



(a) A non-symmetric drawing

(b) A symmetric drawing

Figure 2.16: Two drawings of the same graph considering the symmetry criterion.

- **Area minimization**

  If the used *area* of the drawing is large, then we have to use many pages, or we must reduce the resolution, so either way the drawing becomes unreadable. Therefore, one major objective is to ensure a small area. Minimizing the area of a layout is again crucial for VLSI schematics, but it is also a general aesthetics criterion: *a picture looks much better if the vertices and edges fill the space with homogeneous density.* The ability to construct area-efficient drawings is essential in some practical visualization applications, where saving screen space is so important, see, for example, [PL95, MDV06, YYM12]. This aesthetic is meaningful only if the drawing convention adopted prevents drawings from being arbitrarily scaled down (e.g., grid drawing, or straight-line drawing where any two vertices have distance at least one).

  The area of a drawing can be formally defined in different ways. For example, we can define it as the area of the smallest convex polygon covering the drawing (convex hull), or as the area of the smallest rectangle with horizontal and vertical sides covering the drawing. In Figure 2.18 we give two drawings of the same graph. The drawing in Figure 2.18(b) has a smaller area than the drawing in Figure 2.18(a).

- **Crossing minimization**

  If too many edges cross each other, the human eye cannot easily find out which vertices are connected by an edge. If a graph can be drawn without edge crossings (i.e. planar), then this is very often preferable to a drawing with edge crossings. Crossing minimization is also an important technical criterion. In circuit schematics, wire crossings should be avoided as much as possible to reduce the number of layers.

  In Figure 2.17 we give two drawings of the same graph. The drawing in Figure 2.17(a) has many edge crossings and it is a difficult drawing to follow and understand, where the drawing in Figure 2.17(b) has no edge crossings, which is more pleasing and easy to understand.

(a) A drawing with many crossings (non-planar drawing)

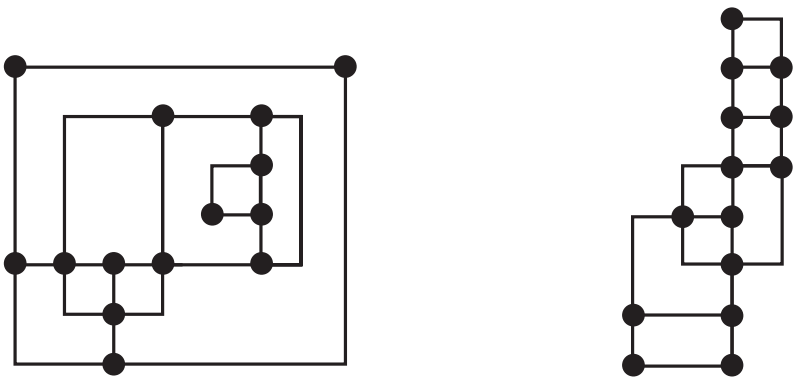
(b) A drawing without crossings (planar drawing)

Figure 2.17: Two drawings of the same graph considering the edge crossings criterion.

- **Edge bends minimization**

  At a *bend*, the drawing of the edge changes direction, and hence a bend on an edge increases the difficulties of following the course of the edge. So, both the total number of bends and the number of bends per edge should be kept small. This is an important aesthetics criterion for orthogonal layouts because the human eye can much more easily follow an edge with none or only a few bends than an edge widely zigzagging through the picture. In VLSI production, bends in wires are potential spots of trouble, so minimizing bends is also an important technical criterion.

  Figure 2.18 represents two orthogonal drawings of the same graph. Figure 2.18 shows a drawing with 8 edge bends while the drawing in 2.18 has only 5 edge bends which is the minimum number of edge bends.



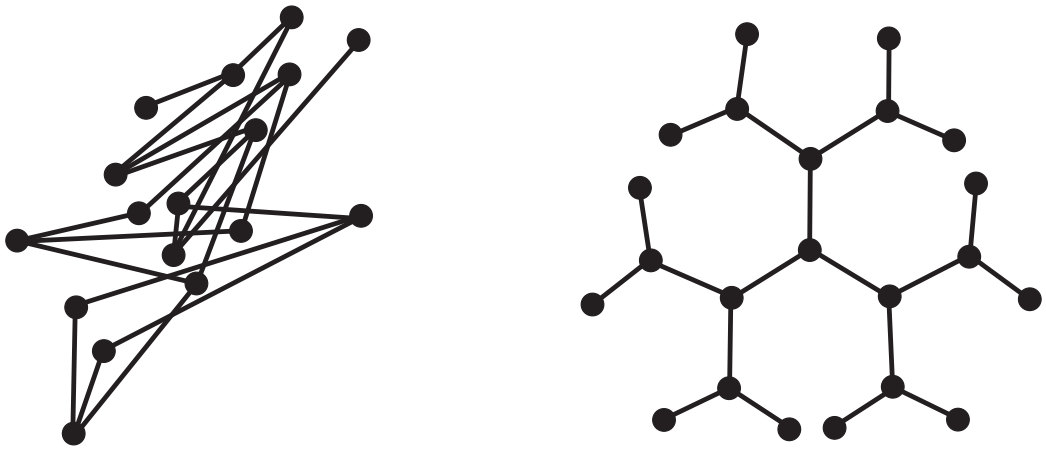
(a) An orthogonal drawing

(b) An orthogonal drawing with minimum number of bends

Figure 2.18: Two orthogonal drawings of the same graph considering the number of bends and area criteria.

- **Angular resolution maximization**

  Angular resolution is measured by the smallest angle between adjacent edges in a drawing. Higher angular resolution is desirable for displaying a drawing on a raster device. The angular resolution aesthetic is especially relevant for straight-line drawings, see [DEG+11, ELMN11, HS12]. See the example presented in Figure 2.19. Some real needs for this aesthetic criterion are: if a graph is displayed on a video (screen with low resolution), it is required that the edges are as far apart as possible. In numerics, simulations using finite element nets behave better if the net drawing have large angle values.



(a) A drawing with poor angular resolution

(b) A drawing of the same graph in (a) with a good angular resolution [LY05]

Figure 2.19: Two drawing of the same graph considering the angular resolution criterion.

- **Aspect ratio**

  The *aspect ratio* of a drawing is defined as the ratio of the length of the longest side to the length of the shortest side of the smallest rectangle with horizontal and vertical sides covering the drawing. A drawing with a high aspect ratio may not be conveniently placed on a workstation screen, even if it has modest area. Hence it is important to keep the aspect ratio small ideally and to obtain small area for any aspect ratio in a given range. This would provide the flexibility of fitting drawings into arbitrarily shaped windows.

- **Edge length minimization:**

  In transportation networks (like railways networks), the length of roads (edges) should be kept as minimal as possible. Also, in VLSI schematics, edges correspond to wires which carry information from one point on the chip to another. To do this fast, wires should be short.

- **Uniform spatial distribution of the vertices**

  Distributing the vertices of the graph uniformly over the area of the drawing considers an aesthetically pleasing drawing that readily and prettily conveys the graph structure.

The above-mentioned aesthetics are naturally associated with optimization problems. However, most of these problems are computationally difficult. So, many approximation strategies, heuristics, and meta-heuristics have been devised and actually applied.

2.4 Summary

This chapter introduces basic concepts in graph theory and graph drawing and its main intention is to provide some essential and notational conventions. The first section gives some general notation about graphs that is also valid for more enhanced models as directed, planar graphs, and hierarchical graphs. The second section deals with the basic style in graph drawing, like orthogonal, straight-line, and upward drawing. Finally, the most well-known drawing aesthetic pleasing criteria and properties are treated in the last section. The notations of this chapter are mainly taken from [DETT94, DETT99, FH01, NR04].

3

Hierarchical Drawing of Directed Graphs

Every block of stone has a statue inside it and it is the task of the sculptor to discover it.[1]

Directed graphs are widely used in applications to model dependency relationships between objects. Examples of these applications include PERT diagrams, organizational charts, VLSI circuits layouts, and subroutine-call graphs, (for more details about the applications, see [DETT99, Chapter 9], [KW01, Chapter 5], [Sug02, Chapter 6], [NR04, Chapter 1]). Directed acyclic graphs are usually represented with polyline (or straight-line if possible) downward (or upward) drawing convention.

This chapter presents the standard hierarchical approach, named Sugiyama framework [STT81], for producing straight-line drawings of directed graphs. In a hierarchical drawing approach, the vertices are placed on parallel horizontal lines and the edges are drawn, keeping a uniform geometric direction, e.g., from top to bottom. Then the final drawing visualizes a common direction of information flow stored by the structure of the original input graph. This approach is highly intuitive and can be applied to any directed graph. In the area of graph drawing, Sugiyama framework is one of the most commonly used drawing methods for graphs.

[1] Michelangelo.

The Sugiyama framework consists of four main steps, where each step is considered a standalone problem and received a lot of attention. In this chapter, we introduce these four steps as optimization problems, the state-of-the-art solution techniques.

Furthermore, the general way of using random hierarchical graphs is to generate random directed graphs and then transfer these generated graphs to some layering technique to get the hierarchical form of these graphs. This way of generating hierarchical graphs is not accurate since there is a gap in the edge density ratio of the generated graphs and their hierarchical ones. We introduce the idea of generating random hierarchical graph directly instead of starting with directed graphs. The proposed techniques for generating random hierarchical graphs covers the gap of edge density.

The chapter is organized as follows. A brief introduction to the Sugiyama approach is given in Section 3.1. The four steps of the Sugiyama approach, including the definition of each problem beside the most considered solution techniques are introduced in Sections 3.2, 3.3, 3.4, and 3.5. The final section, Section 3.7, contains the ideas of the need to generate random hierarchical graphs and also the proposed algorithms for the generation phase.

3.1 Sugiyama Approach

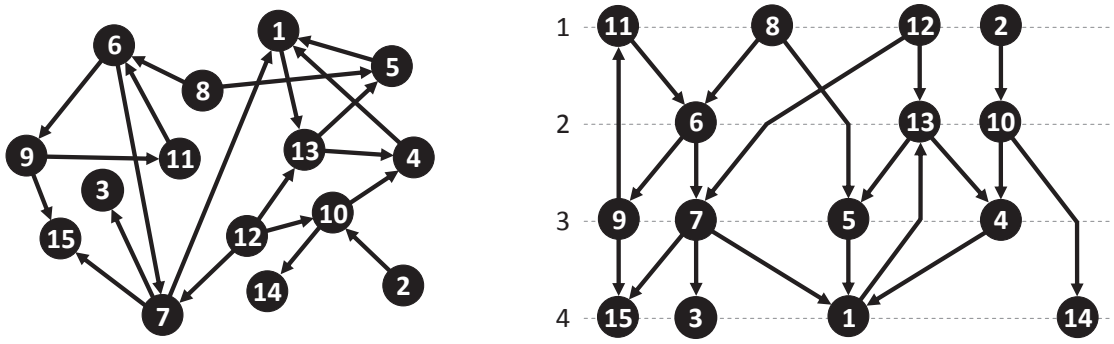
The first attempts on drawing directed graphs in hierarchical form are represented by Warfield [War77] and Carpano [Car80]. Nevertheless, the most common and well-known approach for hierarchical drawings of directed graphs, level graphs, and general arbitrary directed graphs is presented in 1981 by Sugiyama, Tagawa and Toda [STT81] and named the *Sugiyama approach*. Several subsequent methods by Gansner, Koutsofios, North, and Vo [GKNV93], Eades and Sugiyama [ES91], Messinger, Rowe, Lawrence, and Henry [MRH91], Paulisch and Tichy [PT90], Gschwind and Murtagh [GM89], Gansner, North, and Vo [GNV88], and Messinger [Mes88], are closely related.

The Sugiyama approach and its subsequent methods are highly intuitive and can be applied to any directed graph, regardless of its graph-theoretic properties. Thus, they are attractive in practice, and variations of them may be found in several existing systems. Figure 3.1 shows an example of drawing directed graphs in a hierarchical form using the Sugiyama approach.

As it has been mentioned in Chapter 2, we re-assume that the hierarchical graphs have an overall flow or direction from top to bottom. This will be emphasized by drawing most of the edges in one specific direction. Also, edges will be represented as straight-line segments and long edges will be represented as a polyline (or straight-line).

Furthermore, the drawing should achieve some readability aesthetic criteria such as:

1. Edge crossings should be kept as few as possible.
2. Edges with upward direction should be avoided.
3. Long edges should be avoided as possible.
4. Vertices should be uniformly distributed on their layers line.
5. Dummy vertices should be vertically aligned.
6. Aspect ratio of the drawing area should be reasonable.
7. The drawing area should be minimized.



(a) (b)

Figure 3.1: A directed graph drawing (a) and its hierarchical drawing using the Sugiyama approach (b).

It is important to notice that achieving all of the above aesthetic criteria together in one drawing is generally impossible, because some of them are in conflict with each other. For example, aligning the dummy vertices of long edges vertically may increase the drawing width and consequently the area. Moreover, it is also very difficult to produce a drawing that satisfies some of these criteria simultaneously.

The Sugiyama approach for producing hierarchical drawing of directed graphs consists of the following main four steps:

- **Step 1: Cycle Removal**

  Firstly, the algorithm receives a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as input. Then, as few edges as possible that make the graph acyclic are temporarily reversed to make the graph acyclic. This allows to draw all edges in one direction (downward) which is very important for the next step. At the end of the algorithm, the reversed edges are reversed again in order to obtain their original orientation. Techniques for removing cycles in directed graphs are discussed in Section 3.2.

- **Step 2: Layer Assignment**

  In this step, the algorithm receives as input a directed acyclic graph (dag) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ after the cycle removal step. The set of vertices $\mathcal{V}$ of are partitioned into a finite set of *layers* $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k$, such that, if (u, v) is an edge with $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$, then $i < j$. Next, the layered graph is transferred into a *proper layered graph* such that, if (u, v) is an edge with $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$, then $i = j + 1$. This is done by inserting dummy vertices along the edges that span more than two layers, i.e. $i < j - 1$ for edge (u, v), $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$. In the final drawing, the set of vertices in layer $\mathcal{V}_i$ will have the same y-coordinate equals to i. The output of this step is a *layered hierarchical graph* $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$. This step is described in details in Section 3.3.

- **Step 3: Crossing Minimization**

  The proper layered hierarchical directed graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$ produced in the layer assignment step is the input for this step. Since the orders of the vertices on the layers determine the topology of the final drawing, these orders are computed in such a way that the number of crossings is kept as small as possible. The output of this step is a new proper layered directed graph in which an order is specified for the vertices in each layer. Some of the well-known algorithms used for minimizing crossings in hierarchical graphs, including existing techniques and a new proposed efficient barycenter algorithm, are represented in details in Section 3.4.

- **Step 4: Horizontal Coordinate Assignment**

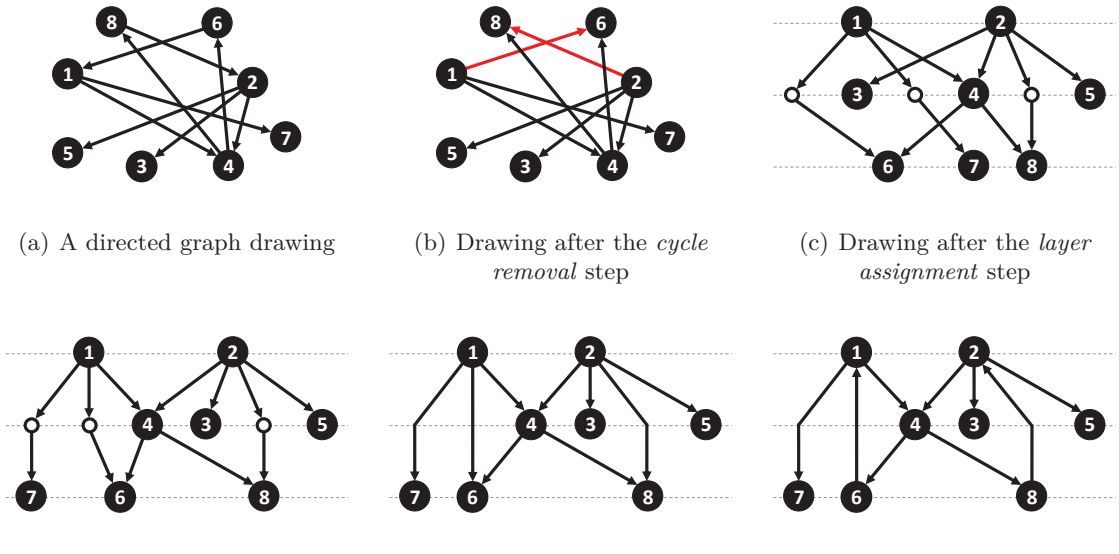
  In this step, the algorithm receives the proper layered hierarchical directed graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$, produced in the previous step, as input where the final x-coordinate $x(v)$ for every vertex $v \in \mathcal{V}_i$, $i = 1, 2, \cdots, k$ are computed. The x-coordinates of the vertices have actually been computed in the layer assignment step. The x-coordinates are computed w.r.t. the ordering determined in the crossing minimization step. Then, each edge is represented as a straight-line segment. Since dummy vertices have no geometrical representation, each long edge may be represented as polyline or polygonal line. In this step, several aesthetic criteria could be taken into account. For example, aligning the dummy vertices of a long edge reduces the number of bends in the final drawing, or displacing the vertices horizontally emphasizes symmetrical drawing of the graph, and also the area of the drawing could be minimized if the vertices could be packed. Some algorithms for this step are represented in Section 3.5.

It is not always necessary to perform all the four steps of the Sugiyama approach. In some cases, the given graph is directed acyclic, then there is no need to implement the cycle removal step and the approach starts directly with implementing the layer assignment step. In other cases, the layering is given together with the graph, i.e. the given graph is a directed acyclic layered graph, then only the last two steps have to be implemented. Furthermore, the last two steps in many cases are combined and implemented together.

The Sugiyama approach could also be applied to undirected graphs by representing arbitrary direction to each edge and then transferring this directed graph to the four steps. Finally, the edges directions are removed from the final drawing keeping the graph edges undirected as given in the original representation of the graph. A detailed example showing the output drawing after each step of the Sugiyama approach is shown in Figure 3.2.

3.2 Cycle Removal

In many applications, such as dependency graphs, the input directed graphs are acyclic, where in other applications the input graph could contain cycles. So, we first have to obtain acyclic directed graph by reversing those edges that make cycles in the directed graph. Then we transfer the graph to the next steps of the Sugiyama approach. Lastly, the graph is rendered with the reversed edges to point in their original direction.



(a) A directed graph drawing

(b) Drawing after the *cycle removal* step

(c) Drawing after the *layer assignment* step

(d) Drawing after the *crossings minimization* step

(e) Drawing after the *horizontal coordinates assignment* step

(f) Final drawing of the original graph after retrieving the direction of the reversed edges

Figure 3.2: Example of the Sugiyama approach. A drawing of the directed graph in (a) and the drawings produced after each step of the Sugiyama approach (b)-(e). Dummy vertices are represented in unfilled small circles

The set of edges whose removal from a directed graph makes it acyclic is known as a *feedback arc set* (*FAS*). The problem of reversing a minimum set of edges is known as *feedback arc set problem*: find a minimum set $\mathcal{E}_f \subset \mathcal{E}$ such that the graph $(\mathcal{V}, \mathcal{E} \setminus \mathcal{E}_f)$ contains no cycles. An alternative problem is the *maximum acyclic subgraph problem*: find a maximum set $\mathcal{E}_a \subset \mathcal{E}$ such that the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}_a)$ contains no cycles. Unfortunately, both problems are $\mathcal{NP}$-hard [Kar72, GJ79] and thus, efficient heuristics are needed. In the following subsections, we present some algorithms for the cycle removal problem.

3.2.1 Vertices Ordering Heuristics

Suppose the set of vertices $\mathcal{V}$ of a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ has an ordering $o : \mathcal{V} \to \{1, 2, \cdots, |\mathcal{V}|\}$. The easiest heuristic for the maximum acyclic subgraph problem is to get an arbitrary ordering of the graph vertices and then delete the edges (u, v) with $o(u) > o(v)$. It is possible to use a given ordering, or use an ordering computed by applying breadth first search [BJG08] or depth first search [BJG08] to the graph. These heuristics do not allow any quality guarantees to be given, but they are fast [BM01].

Another fast and simple greedy heuristic (Algorithm 3.1) is introduced by Berger and Shor [BS90]. The idea of this heuristic is to delete for every vertex $v \in \mathcal{V}$ either its set of incoming edges $N^-(v) = \{(u, v) \mid (u, v) \in \mathcal{E}\}$ or the outgoing ones $N^+(v) = \{(v, w) \mid (v, w) \in \mathcal{E}\}$, $N(v) = N^-(v) \cup N^+(v)$. Taking always the smaller set of $N^-(v)$ and $N^+(v)$ leads to an acyclic set FAS with size $|FAS| \leq \frac{1}{2}|\mathcal{E}|$. The greedy heuristic runs in linear time of $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}|)$.

Algorithm 3.1: Greedy Heuristic Cycle Removal

Input : A directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
Output: A vertex sequence $\mathcal{S}$ of $\mathcal{G}$;

```
1 S = ∅;
2 foreach v ∈ V do
3     if |N+(v)| ≥ |N−(v)| then
4         append N+(v) to S;
5     else
6         append N−(v) to S;
7     delete N(v) from G;
```

Based on the observation that edges incident edges to source or target vertices cannot be part of a cycle, an enhanced greedy heuristic (Algorithm 3.2) is introduced [ELS93]. The only difference between the greedy heuristic and the enhanced greedy heuristic is that the enhanced one processes the vertices in a special order. Hence, the output of the enhanced greedy heuristic is acyclic as well and it computes an acyclic edge set FAS with an upper bound $|FAS| \leq \frac{|\mathcal{E}|}{2} + \frac{|\mathcal{V}|}{6}$ and it needs $\mathcal{O}(|\mathcal{E}|)$ running time.

Sander [San99] suggested a more elaborate heuristic but similar version heuristic of the enhanced greedy heuristic. In the proposed heuristic, we choose the next vertex $v \in \mathcal{V}$ based on strongly connected components. Although promising practical results are reported, however, the computational time is still with the same complexity $\mathcal{O}(|\mathcal{E}| \cdot |\mathcal{V}|)$ [Bac09].

Algorithm 3.2: Enhanced Greedy Heuristic Cycle Removal

Input : A directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
Output: A vertex sequence $\mathcal{S}$ of $\mathcal{G}$;

```
1  S = ∅;
2  while G is not empty do
3      while G contains a sink v do
4          add N−(v) to S;
5          delete v and N−(v) from G;
6      while G contains a source v do
7          add N+(v) to S;
8          delete v and N+(v) from G;
9      if G is not empty then
10         let v be a vertex in G with a maximum value |N+(v)| − |N−(v)|;
11         add N+(v) to S;
12         delete v and N(v) from G;
```

3.2.2 Randomized Heuristic

Berger and Shor [BS90] presented a randomized version of the greedy heuristic (Algorithm 3.1), which is the randomized greedy heuristic. The only difference is that: in the randomized version, the vertices are ordered randomly at the beginning. They proved that the expectation value $|FAS| \leq \frac{1}{2} + \Omega\left(\frac{1}{\sqrt{deg(\mathcal{G})}}\right) \cdot |\mathcal{E}|$ is valid as the worst-case bound on $|FAS|$ for their deterministic heuristic.

3.2.3 Cycle Breaking Heuristic

Another way of solving the minimum FAS problem is to build the FAS edge by edge and choosing those edges that create cycles, instead of focusing on computing linear orderings of the vertices which provide FAS. A simple heuristic based on this idea is: starting with two empty sets S and T, we scan all edges one by one. For each edge $e \in \mathcal{E}$, if $S \cup \{e\}$ is acyclic, then e is added to S, otherwise e will be added to T. At the end of this process, it is clear that both sets S and T are acyclic, and the smaller one of S and T contain an FAS which has at most half of the edges of the directed graph. Note that T is a minimal FAS, while S might not be.

In the dot system [GKNV93], Gansner, Koutsofios, North, and Vo have introduced a heuristic related to this approach. It takes one non-trivial strongly connected subgraph component of the directed graph at a time, in some arbitrary order. Within each component it performs a depth-first traversal and adds to the FAS an edge which participates in a maximum number of cycles. This is repeated until there are no more non-trivial strongly connected components. It has been reported in [GKNV93] that this heuristic works well in the experiments. The best-known approximation algorithm achieves a performance ratio $\mathcal{O}(\log|\mathcal{V}| \log\log|\mathcal{V}|)$ and is presented by Even, Naor, Rao, and Schieber [ENRS00].

3.2.4 Exact Algorithm

For an exact Integer Linear Programming (ILP) approach to the minimum FAS problem, Saab proposed a divide-and-conquer algorithm [Saa01]. Also, Grötschel, Jünger and Reinelt [GJR85] and Rienelt [Rei85] introduced a study of the facial structure of the acyclic subgraph polytope which can be used for finding the minimum FAS by a branch-and-cut algorithm.

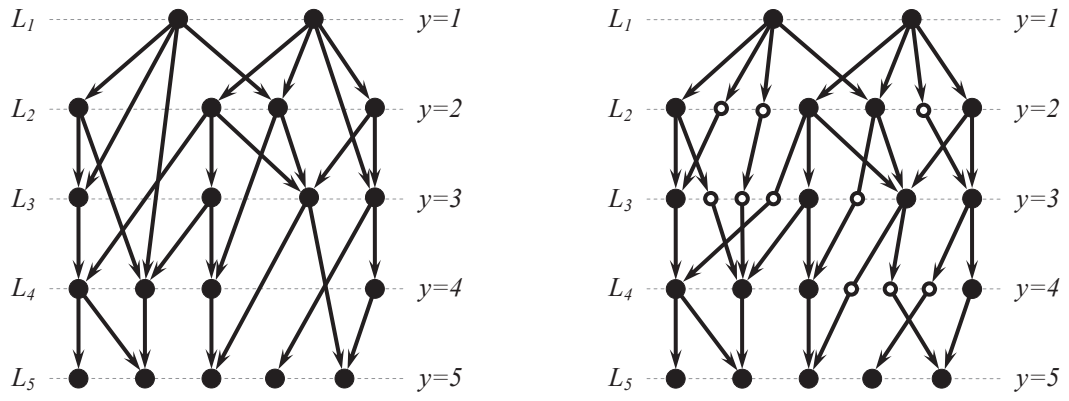
3.3 Layer Assignment

The main goal of this step is to assign a y-coordinate to each vertex in an directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. A *layering* ϕ of $\mathcal{G}$ is a partitioning of the set of vertices $\mathcal{V}$ into a finite number k of subsets (called layers) $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k$, i.e. $\mathcal{V} = \bigcup_{i=1}^{k} \mathcal{V}_i$, such that the vertices of every layer $\mathcal{V}_j, 1 \leq j \leq k$, are placed on a horizontal line $L_j = \{(x, k-j) \mid x \in R\}$, and every edge $(u, v) \in \mathcal{E}$, $u \in \mathcal{V}_i$, $v \in \mathcal{V}_j$, $1 \leq i < j \leq k$, is drawn as a monotone decreasing curve (normally straight-line segment) between the lines L_i and L_j, as shown in Figure 3.3. The

produced *hierarchical* graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$ is also called a *k-layered* directed graph and could be represented as $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$. The *span* $s(e)$ of an edge $e = (u, v)$ with $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ is $j - i$ or $y(v) - y(u)$. If no edge in the hierarchical has a span greater than one then the hierarchical graph is *proper*.

There are three important aesthetic criteria requirements which should be kept when a layering method is considered:

1. *The hierarchical graph should be compact.* Compactness can be achieved by minimizing the width and the height of the graph. The *width* of a hierarchical graph is the number of vertices in the *longest layer*, that is, $max_{1 \leq i \leq k} |\mathcal{V}_i|$, and the *height* is the number of layers k. A simple algorithm to compute a layering with minimum height is given in Section 3.3.1 and another one for compute a layering within a given width is given in Section 3.3.2. Unfortunately, finding a layering by minimizing the height with respect to a given width is $\mathcal{NP}$-hard [CG72, GJ79].
2. *The hierarchical graph should be proper.* This can be easily achieved by introducing dummy vertices into the layering for every long edge (u, v) with $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ where $i < j - 1$. In other words, we replace the long edge (u, v), $u \in \mathcal{V}_i$, $v \in \mathcal{V}_j$, with the path $(u, v_1, v_2, \cdots, v_l, v)$ where a dummy vertex v_p, $i + 1 \leq p \leq j - 1$, is inserted in each intermediate layer $\mathcal{V}_{i+p}$. Figure 3.3 shows an example of breaking long edges using dummy vertices. Most of the algorithms used in the subsequent steps of the Sugiyama approach need to have proper layering.
3. *The number of dummy vertices should be kept minimum.* There are three reasons for minimizing the number of dummy vertices. Firstly, the running time (of most of the algorithms) of the next steps of the Sugiyama approach depends on the total number of vertices including the dummy ones. Secondly, bends in the drawing will only occur at dummy vertices, so a small number of dummy vertices means a small number of edge



(a) (b)

Figure 3.3: An example of adding dummy vertices. The hierarchical drawing in (a) contains long edges, where these long edges have been broken in drawing (b) by adding dummy vertices (drawn as small empty circles) to the traversed layers.

bends which is considered as a required aesthetic drawing criterion to increase the final drawing readability. Finally, the edges will become long if many dummy vertices occur. An algorithm for computing a layering with minimizing the number of dummy vertices is presented in Section 3.3.3.

3.3.1 Minimizing The Height

This algorithm requires that the input directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to be acyclic. It computes a layering with minimum height by applying the longest path method [BJG08]. Firstly, each source vertex is placed in the first layer $\mathcal{V}_1$. Then, the layer $\phi(v)$ for every remaining vertex v is recursively defined by $\phi(v) = \max\{\phi(u) \mid (u, v) \in \mathcal{E}\} + 1$. This algorithm produces a layering where many vertices will stay close to the bottom, and hence the number of layers k is kept minimized. By using a topological ordering of the vertices [Meh84], the algorithm can be implemented in linear time $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$. The main drawback of this algorithm is that it may produce drawings that are too wide.

3.3.2 Layering with Given Width

The longest path layering algorithm minimizes the height, while compactness of the final drawing depends on both the width and the height. The problem of finding a layering with minimum height is $\mathcal{NP}$-complete if a fixed width greater or equal to three is given [GJ79]. The Coffman-Graham algorithm [CG72] considers a layering with a maximum width. Minimizing the number of dummy vertices guarantees minimum height [ES91]. The Coffman-Graham algorithm takes as input a reduced graph, i.e., no transitive edges are included in the graph, and a given width w. An edge (u, v) is called *transitive* if a path $(u = v_1, v_2, \cdots, v_k = v)$ exists in the graph.

The Coffman-Graham algorithm works in two phases. The first orders the vertices by their distance from the source vertices of the graph. In the second phase, vertices are assigned to the layers, such that vertices with large distances from the sources will be assigned to layers as close to the bottom as possible.

Lam and Sethi [LS77] showed that the number of layers k of the computed layering with width w is bounded by $k \leq \left(2 - \frac{2}{w}\right) \cdot k_{opt}$, where k_{opt} is the minimum height of all layerings with width w. So, the Coffman-Graham algorithm is an exact algorithm for $w \leq 2$. The notion of width does not consider dummy vertices, since dummy vertices have very small width (usually the thickness of an edge line segment) in comparison to real vertices which contain normally significant text strings. Even though the Coffman-Graham is currently the most commonly used layering method [Bac09].

3.3.3 Minimizing the Total Edge Span

The objective to minimize the total edge span (or edge length) is equivalent to *minimizing the number of dummy vertices*, which is a reasonable objective in the final drawing. It can be shown that minimizing the number of dummy vertices guarantees minimum height [ES91]. Gansner, Koutsofios, North, and Vo [GKNV93] introduced the network simplex algorithm, which is the first attempt to generate layerings with the minimum number of dummy vertices by modelling the problem as an ILP system (Equation 3.1).

Algorithm 3.3: Coffman-Graham Layering

Input : A directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a positive integer w
Output: Layering of $\mathcal{G}$ of width at most w

foreach $v \in \mathcal{V}$ **do**
  $\phi(v) = n + 1$;
for $i = 1$ *to* $|\mathcal{V}|$ **do**
  Choose a vertex v with $\phi(v) = n + 1$ and minimum set $\{\phi(u) \mid (u, v) \in \mathcal{E}\}$ with respect to $\prec$;
  $\phi(v) = i$;
$k = 1$; $\mathcal{V}_1 = \emptyset$; $U = \mathcal{V}$;
while $U \neq \emptyset$ **do**
  choose $u \in U$, such that every vertex in $\{v \mid (u, v) \in \mathcal{E}\}$ is in $\mathcal{V} \setminus U$ and $\phi(u)$ is maximized;
  if $|\mathcal{V}_k| < w$ and for every edge (u, w), $N^+(u) \subseteq \mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_{k-1}$ **then**
    add u to $\mathcal{V}_k$;
  else
    $k := k + 1$; $\mathcal{V}_k := \{u\}$;
  delete u from U;

$$\text{span}(\mathcal{G}) := \min \sum_{(u,v)\in\mathcal{E}} (\phi(v) - \phi(u)) \tag{3.1}$$

subject to

$$\begin{aligned} \phi(v) - \phi(u) &\geq 1 \quad \text{for} \quad (u, v) \in \mathcal{E} \\ \phi(v) &\in \mathbb{N} \quad\quad\; \text{for} \quad v \in \mathcal{V} \end{aligned}$$

Standard linear programming could solve this ILP problem and find an optimal solution since the constraints matrix is totally unimodular [NW88]. The network simplex algorithm has been proved to be very efficient experimentally, even though it does not guarantee a polynomial running time.

Another branch-and-cut approach was introduced by Healy and Nikolov [HN02]. This branch-and-cut approach finds layerings with the minimum number of dummy vertices subject to upper bounds on both the width and the height of the layering if there is any feasible solution. Nikolov and Tarassov [NT06] showed that layerings found by the longest-path and Coffman-Graham algorithms can be easily improved by a simple vertex-promotion heuristic.

3.3.4 Minimum Width

Practically, minimizing the number of dummy vertices in a hierarchical graph not only gives shorter edge lengths and fewer dummy vertices, but also gives relatively compact layerings. Combining the minimization of the height of a drawing with the minimization of the number of dummy vertices is $\mathcal{NP}$-complete [Lin92].

MinWidth and StretchWidth are two algorithms proposed by Branke, Nikolov, and Tarassov [TNB04, NTB05] to solve the $\mathcal{NP}$-hard problem of minimum-width layering of a directed acyclic graph considering the dummy vertices. An earlier attempt is the heuristic developed by Branke, Leppert, Middendorf, and Eades [BLME02].

The MinWidth heuristic [NTB05] is based on the longest path algorithm that is shown in Section 3.3.1. Two variables widthCurrent and widthUp are employed to keep the width of the current layer, and the width of the layer above it, respectively. The current layer width widthCurrent, is calculated as the summation of the number of original vertices already assigned to that layer and the number of potential dummy vertices along edges with a source in $\mathcal{V} \setminus U$ and a target in Z (one dummy vertex per edge). An estimation of the width of any layer above the current one is saved in the variable widthUp, which is the number of potential dummy vertices along edges with a source in $\mathcal{V} \setminus U$ and a target in the current layer (one dummy vertex per edge). When a vertex is selected to be assigned to a layer, an additional condition ConditionSelect is used, which is True if v is the vertex with the maximum outdegree among the candidates to be added to the current layer. Such a choice of v results in maximum reduction to widthCurrent. The Minwidth layering algorithm has a worst-case time complexity $\mathcal{O}(|\mathcal{V}| \log |\mathcal{V}| + |\mathcal{E}|)$.

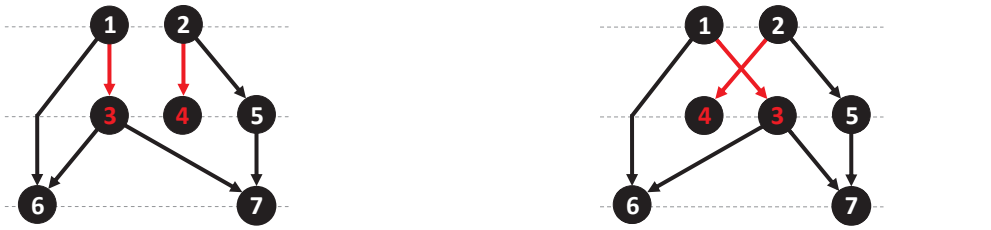
StretchWidth algorithm builds the layering by trying to have its width lower than or equal to an upper bound, which gradually gets bigger. Initially, we set up the upper bound at $\max\{\max\{d^+(v) : v \in \mathcal{V}\}, \max\{d^-(v) : v \in \mathcal{V}\}\}$. If the algorithm reaches a point where it is impossible to assign a vertex to a layer without going above the upper bound, the upper bound is incremented by one and start over. StretchWidth can be implemented in a total worst-case time complexity $O(|\mathcal{E}||\mathcal{V}| \log |\mathcal{V}| + |\mathcal{E}|^2)$.

Promote layering [NT06] is a heuristic whose goal is "*to develop a simple and easy to implement layering method for decreasing the number of dummy vertices in a DAG layered by some list scheduling algorithm.*" The promote layering method is an alternative to the network simplex [GKNV93] (presented in Section 3.3.3) but considerably easier to implement and especially useful when a commercial linear programming solver is not available [AHN07]. It has been mentioned in [NTB05] that the algorithm MinWidth, followed by the vertex promotion heuristic produces a layering with the minimum width considering dummy vertices. The *ant colony* methaeuristic has been applied by Andreev, Healy and Nikolov [AHN07] in order to get a layering for directed acyclic graphs.

3.4 Crossing Minimization

This section concerns the problem of drawing hierarchical graphs with minimum number of crossings between edges. Edge crossings, for readability of drawings, are crucial parameters that should be considered. In her experimental studies [Pur97, Pur02], Purchase mentioned that minimizing the number of edge crossings is a major parameter for supporting an easy human understanding of graph drawing. Crossing minimization is also of special consideration to VLSI layout researchers and has a history that progressed much of the graph drawing literature [War77, EK86, EW94a, EW94b, Cat95, VML96, JM97, JLMO97, LMV97, MSM99, LM99, Mut00, GSBM01, YT00, YT01, EGDB02, MUV02, KPL06, CGMW10, BBBH11, BBG11, CGMW11]. We assume that the input for the crossing minimization step is a proper layered hierarchical graph.

Suppose, $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in \mathcal{E}$ are two short edges in a hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$ such that $u_1, u_2 \in \mathcal{V}_i$, $v_1, v_2 \in \mathcal{V}_{i+1}$, $1 \leq i \leq k-1$, then e_1 and e_2 cross each other if $(\pi(u_2) - \pi(u_1)) \cdot (\pi(v_2) - \pi(v_1)) < 0$ where $\pi(v)$ is the order of vertex v in its layer. See the example given in Figure 3.4. Counting the number of edge crossings and its bounds of a graph drawing has received a lot of attention in the area of graph drawing (see [War77, GP83, SSV95, SV97, PT00, HS07, CM11]).



(a) $\pi(1) = 1, \pi(2) = 2, \pi(3) = 1$ and $\pi(4) = 2$ (b) $\pi(1) = 1, \pi(2) = 2, \pi(3) = 2$ and $\pi(4) = 1$

Figure 3.4: Example of computing edge crossing according to the relative order of the vertices in their layers. In drawing (a) no crossing between the two edges (1,3) and (2,4) since $(\pi(2) - \pi(1)) \cdot (\pi(4) - \pi(3)) = 1 \cdot 1 = 1 > 0$. In drawing (b), the two vertices 4 and 5 are switched in their order and hence the same two edges cross since $(\pi(2) - \pi(1)) \cdot (\pi(4) - \pi(3)) = 1 \cdot (-1) = -1 < 0$.

An important observation here is that the number of edge crossings in a drawing of a hierarchical graph does not depend on the precise position (x-coordinates) of the vertices in their layers, but only on the ordering of the vertices within each layer. Hence, the problem of minimizing edge crossings in a hierarchical graph is a combinatorial one of choosing an appropriate vertex ordering for each layer, not a geometric one of choosing an x-coordinate for each vertex. Although this combinatorialization simplifies the problem, it is still difficult and has a classification of $\mathcal{NP}$-hard even with a graph having just two layers (i.e. bipartite graph) [GJ83]. Furthermore, the crossing minimization problem still remains $\mathcal{NP}$-hard even if one layer has a fixed vertex ordering [EW94a, EW94b] and for sparse graphs with vertex degree equals to 4 [MUV02].

We assume that the directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is actually converted into the proper k-layer hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$. This means $\mathcal{V} = \mathcal{V}_1 \cup V_2 \cup \cdots \cup V_k$ such that $\mathcal{V}_i \cap V_j = \emptyset, 1 \leq i \neq j \leq k$ and all the edges in $\mathcal{E}$ are represented as *short* ones, i.e., $\mathcal{E} = \{(u, v) \mid u \in \mathcal{V}_i, v \in \mathcal{V}_{i+1}, 1 \leq i \leq k-1\}$. We let $n = |\mathcal{V}|$, $n_i = |\mathcal{V}_i|, m = |\mathcal{E}|$, and the neighbourhood $N(v)$ of vertex v is $N(v) = N^-(v) \cup N^+(v)$ such that $N^-(v) = \{u \in \mathcal{V} \mid (u, v) \in \mathcal{E}\}$ and $N^+(v) = \{w \in \mathcal{V} \mid (v, w) \in \mathcal{E}\}$.

An *ordering* (or *permutation*) π_i of the set of vertices belonging to layer $\mathcal{V}_i$, $1 \leq i \leq k$ in a hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$ provides a solution for the crossing minimization problem since it is the relative ordering along the line $L_i = i$ that causes edges incident on that layer to cross each other. What we search for is the set of permutations, $\prod = \{\pi_i \mid 1 \leq i \leq k\}$ that minimizes the edge crossings $cross(\mathcal{H}, \pi_i, \pi_{i+1})$, $1 \leq i \leq k$.

A rich variety of heuristics is used to minimize crossings in hierarchical graphs. Many heuristics have been developed to minimize edge crossings but only some work has been done on minimizing the crossings in the whole graph globally at once. The *layer-by-layer sweep*, presented in Section 3.4.1.1, is the general framework of most techniques. The most critical part of the layer-by-layer sweep is an algorithm for the *two-layer crossing minimization problem*, which is a technique for minimizing crossings between two layers, i.e. bipartite graph. Most of the known techniques for minimizing crossings in the layer-by-layer sweep framework are introduced in Section 3.4.2.

3.4.1 Crossing Minimization Approaches

3.4.1.1 The Layer-by-Layer Sweep Approach

The *layer-by-layer sweep* technique works, in the down sweep, as follows: Firstly, a vertex ordering of the layers is initially chosen, where the vertices get their positions in left-to-right order. In the next step, a layer with a precomputed ordering, e.g., layer $\mathcal{V}_1$, is chosen and for $i = 2, 3, \cdots, k$, the vertex ordering of layer $\mathcal{V}_{i-1}$ is kept fixed while the vertices in layer $\mathcal{V}_i$ are reordered to reduce the crossings between the two layers $\mathcal{V}_{i-1}$ and $\mathcal{V}_i$. We can sweep back from layer $\mathcal{V}_k$ to layer $\mathcal{V}_1$ and repeat these two steps until no further reduction of crossings could be achieved. In the up sweep the roles are switched. This problem is called the *one-sided crossing minimization problem* and will be deeply discussed in the next section.

When using any one-sided crossing minimization algorithm, the number of crossings between the two layers $\mathcal{V}_{i-1}$ and $\mathcal{V}_i$ is reduced by permuting the vertices in layer $\mathcal{V}_i$, while the number of crossings between $\mathcal{V}_i$ and $\mathcal{V}_{i+1}$ could be increased when permuting $\mathcal{V}_i$. These heuristics push the crossings downwards or upwards, according to the direction flow, until they are resolved at layer k or 1, respectively. Other ways of sweeping are possible, for instance we can hold a layer in the middle fixed and sweep from here to the bottom and to the top layer. So, an extension is the *centered 3-level crossing reduction*, i.e., considering the three consecutive layers $\mathcal{V}_{i-1}$, $\mathcal{V}_i$, and $\mathcal{V}_{i+1}$ by permuting $\mathcal{V}_i$ while the orders of $\mathcal{V}_{i-1}$ and $\mathcal{V}_{i+1}$ are fixed such that the crossings between the three layers are reduced. However, the key problem of the layer-by-layer sweep is to reduce the crossings between two layers with the permutation of one side fixed.

In Section 3.4.2, the one-sided two-layer crossing minimization problem and well-known solving techniques are presented. Also, the multi-layer crossing minimization problem is discussed in Section 3.4.3 beside its most-mentioned solving techniques.

3.4.1.2 Global Approach

A main drawback of the layer-by-layer approach is that it may be stuck in a local optimum although the one-sided two-layer crossing reduction returns optimal vertex orderings of the layers in the sense of a minimum number of crossings. Bastert and Matuszewski claimed in [KW01, page 102] that the results are even far from optimum. Alas, potentially existing approximation ratios of one-sided two-layer reduction algorithms cannot be actually extended to be applied to k-layer hierarchical graphs [Bac09].

Although two-layer algorithms reduce the crossings between $\mathcal{V}_{i-1}$ and $\mathcal{V}_i$, the number of crossings between $\mathcal{V}_i$ and $\mathcal{V}_{i+1}$ (and thus even the total number of crossings) can increase while permuting the middle layer $\mathcal{V}_i$. These heuristics push the crossings downwards or upwards until they are resolved at layer k or 1, respectively. Even worse, there may remain type-2 conflicts (cross between two edges that connect only dummy vertices). Alternatives are ordered *k-layer sifting* or the similar *centered three-layer* crossing reduction described in Section 3.4.3. However, both generate many type-2 conflicts and are for reaching a global optimum both restricted to a local view. Thus, they also may tend to get stuck in local optima.

Bachmaier, Brandenburg, Brunner, and Hübner [BBBH11] introduced a *global sifting* algorithm (see Section 3.4.3), that does not use the sweeps with two-layer one side fixed crossing reduction. They consider the block graph − which is the same as their compaction graph completed with one sweep over all layers − directly for crossing reduction. Then the whole graph is treated. Experimentally, global sifting algorithm produced better results than any one-sided two-layer method, it works in higher complexity and has running time bound about $\mathcal{O}(|\mathcal{E}|^2)$ for a hierarchical k-layered hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$.

3.4.2 One-Sided Crossing Minimization

A *bipartite graph* $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ is an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in which $\mathcal{V}$ is partitioned into two sets $\mathcal{V}_1$ and $\mathcal{V}_2$ such that $(u, v) \in \mathcal{E}$ implies either $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$ or $v \in \mathcal{V}_1$ and $u \in \mathcal{V}_2$. An *ordering of layer* $\mathcal{V}_i$ is specified by a permutation π_i of $\mathcal{V}_i$. So, the ordering of the two layers $\mathcal{V}_1$ and $\mathcal{V}_2$ are the permutations π_1 and π_2 respectively.

Let $cross(\mathcal{G}, \pi_1, \pi_2)$ be the number of edge crossings in a straight-line drawing of $\mathcal{G}$ given by π_1 and π_2. If the permutation π_1 of $\mathcal{V}_1$ is kept fixed, the minimum number of edge crossings that could be achieved by reordering the vertices in $\mathcal{V}_2$ is given by $opt(\mathcal{G}, \pi_1)$, where:

$$opt(\mathcal{G}, \pi_1) = \min_{\pi_2} cross(\mathcal{G}, \pi_1, \pi_2)$$

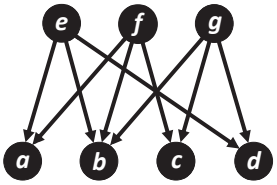
So, using this terminology, the one-sided crossing minimization problem could now be formulated as follows:

> *Given a bipartite graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$ with a permutation π_1 of $\mathcal{V}_1$. Find an ordering π_2 of $\mathcal{V}_2$ that minimizes the edge crossings in the drawing of $\mathcal{G}$, such that $cross(\mathcal{G}, \pi_1, \pi_2) = opt(\mathcal{G}, \pi_1)$.*

The two-layer crossing minimization problem is $\mathcal{NP}$-hard [EW94b] and received a great deal of attention in the graph drawing area, see, for example, [KW01, ch. 5] and [DETT99, ch. 9]. The rest of this section discusses the main heuristics methods for the two-layer crossing minimization problem.

The notion of the crossing number, introduced by Eades and Kelly [EK86], is important for many heuristics. π_1 is the permutation of layer $\mathcal{V}_1$ and is kept fixed, for each pair of vertices $u, v \in \mathcal{V}_2$, c_{uv} represents the number of crossings between edges that incident on u and edges incident on v, when $\pi_2(u) < \pi_2(v)$. Also, for all $u \in \mathcal{V}_2$, we define $c_{uu} = 0$. An observation should be taken into account that the number of crossings between edges incident

on u and edges incident on v depends only on the relative positions of u and v and not on the positions of the other vertices. Figure 3.5(a) shows a drawing of a bipartite graph and its corresponding crossing number matrix for each pair of vertices in the top layer is depicted in Figure 3.5(b).



(a)

	a	b	c	d
a	0	1	0	1
b	3	0	1	2
c	3	3	0	2
d	2	2	1	0

(b)

Figure 3.5: A bipartite graph drawing (a) and its crossing numbers matrix (b).

We can use the crossing numbers to compute $cross(G, \pi_1, \pi_2)$ as follows:

$$cross(G, \pi_1, \pi_2) = \sum_{\pi_2(u)<\pi_2(v)} c_{uv} = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ij}$$

The lower bound of the number of crossings of the two-layer crossing minimization problem is computed as:

$$\sum_{\pi_2(u)<\pi_2(v)} \min\{c_{uv}, c_{vu}\}$$

and the optimal number of edge crossing $opt(\mathcal{G}, \pi_1)$ is computed as:

$$opt(\mathcal{G}, \pi_1) \geq \sum_{u,v} \min(c_{u,v}, c_{v,u})$$

Jünger and Mutzel showed in their experiments [JM97] that this simple lower bound is very tight to the optimum.

We will now consider in details the most interesting heuristics used in the two-layer crossing minimization problem.

3.4.2.1 Barycenter Heuristic

The barycenter heuristic [STT81] is the most common method considered for the two-layer crossing minimization problem since it is very popular, easy to implement, runs fast, and gives good results [KW01]. The barycenter heuristic is based on the intuition that in a drawing with few crossings, each vertex should be close to its adjacent vertices. In this heuristic, for a two-layer hierarchical graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$, we compute the x-coordinate $x(v)$ of a vertex $v \in \mathcal{V}_2$ as the barycenter (average) of the x-coordinates of its neighbours in layer $\mathcal{V}_1$. In order to do this, we compute:

$$x(v) = \frac{1}{deg(v)} \sum_{u \in N(v)} x(u)$$

where $N(v) = \{u \in \mathcal{V}_1 : \{u, v\} \in \mathcal{E}\}$ and $deg(v) = |N(v)|$. If two values are equal we separate them arbitrarily by a small amount. Then the vertices in layer $\mathcal{V}_2$ are sorted by their barycenter values. The barycenter heuristic (Algorithm 3.4) gives a planar drawing if one is possible.

Since the running time for computing $x(v)$ is proportional to the degree of v, the barycenter values of all vertices can be found in linear time of the number of edges $|\mathcal{E}|$. For the subsequent sorting step, the time complexity is $O(|\mathcal{V}_2| \log |\mathcal{V}_2|)$. Hence, the barycenter heuristic runs in $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}_2| \log |\mathcal{V}_2|)$.

Algorithm 3.4: Barycenter Heuristic Crossing Minimization

Input : A bipartite graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ and a vertex ordering π_1 of the vertices in layer $\mathcal{V}_1$

Output: Vertex ordering π_2 of the vertices in layer $\mathcal{V}_2$

```
repeat
    for v := 1 to |V_2| do
        x(v) = 1/deg(v) * sum_{u in N(v)} x(u);
    Sort the vertices in V_2 according to their x-coordinates;
until the number of crossings does not reduce;
```

3.4.2.2 Median Heuristic

The median heuristic [EW94a] is similar to the barycenter heuristic. In the median heuristic, the x-coordinate of each vertex $v \in \mathcal{V}_2$ is given by the *median* of the x-coordinates of its neighbours in layer $\mathcal{V}_1$. Here, the median is defined as follows: suppose the neighbours of v are $u_1, u_2, \cdots, u_j$ with $\pi_1(u_1) < \pi_1(u_2) < \cdots < \pi_1(u_j)$, then $med(v) = \pi_1(u_{\frac{j}{2}})$. This definition differs from the classical notion of the median since if j is even, then there are actually two medians at $\frac{j}{2}$ and $\frac{j}{2} + 1$. Here, we take always the left median. Furthermore, we set $med(v) = 0$, if vertex v has no neighbours.

As with the barycenter heuristic, we have to sort the vertices in layer $\mathcal{V}_2$ according to their median values. If two vertices have the same median they are separated by a small amount, with the restriction that if one vertex has an odd degree and the other vertex has an even degree, then the odd-degree vertex is placed on the left of the vertex with even degree. If the degrees of the vertices have same parity, we can choose their order arbitrarily. The median heuristic (Algorithm 3.5) gives a planar drawing if it is possible.

For each vertex $v \in \mathcal{V}_2$, $med(v)$ can be computed in proportional to the degree of v, and the median could be found in linear-time algorithm (see, for example [AHU83]). For the sorting step, the *bucket sort* technique can be used in order to get a linear-time complexity of $O(|\mathcal{V}_2|)$. Hence, the median heuristic runs in $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}_2|)$. Note, that the crossing number matrix is not required to be precomputed in barycenter heuristic and median heuristic as in several other heuristics.

For a two-layer hierarchical graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ with a fixed ordering π_1 of the layer $\mathcal{V}_1$, the relation between the optimal number of crossing $opt(\mathcal{G}, \pi_1)$ and the number of crossings produced by the median heuristic $med(\mathcal{G}, \pi_1)$ is $med(\mathcal{G}, \pi_1) \leq 3\ opt(\mathcal{G}, \pi_1)$ [EW94a,DETT99]. This result has been proved by Eades and Wormald in [EW94a] and by Di Battista, Eades, Tammassia and Tollis in [DETT99].

Algorithm 3.5: Median Heuristic Crossing Minimization

Input : A bipartite graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ and a vertex ordering π_1 of the vertices in layer $\mathcal{V}_1$
Output: Vertex ordering π_2 of the vertices in layer $\mathcal{V}_2$

```
1 repeat
2     foreach v ∈ V_2 do
3         Sort the vertices in N(v) according to their order π_1;
4         x(v) = π_1(u_⌊j/2⌋) such that u ∈ N(v) and 1 ≤ j ≤ |N(v)|;
5     Sort the vertices in V_2 according to their x-coordinates;
6 until the number of crossings does not reduce;
```

3.4.2.3 Adjacent-Exchange Heuristic

The adjacent-exchange heuristic (Algorithm 3.6), also called greedy switching, works in a way similar to *bubble-sort*. If u and v are two consecutive vertices in $\mathcal{V}_2$, then switching their positions changes the total number of crossings by exactly $c_{vu} - c_{uv}$. The algorithm scans all consecutive pairs of vertices and switches them if this reduces the number of crossings. This process is repeated until no further switching occurs, i.e., for all consecutive pairs (u, v) the inequality $c_{uv} \leq c_{vu}$ holds. Such a vertex ordering is called *stable*. Since one scan of the vertices can be implemented in $O(|\mathcal{V}_2|)$ and there are at most $|\mathcal{V}_2|$ scans, the time complexity of the greedy switching heuristic is $O(|\mathcal{V}_2|^2)$.

Mäkinen [Mäk90] and Gansner, Koutsofios, North, and Vo [GKNV93] suggested that the adjacent-exchange heuristic is preferable as a post-processing step in combination with other heuristics such as barycenter or median heuristic. This is because adjacent-exchange heuristic does not recompute the sorting completely, but makes changes only when it improves the result.

Algorithm 3.6: Adjacent-Exchange Heuristic Crossing Minimization

Input : A bipartite graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ and a vertex ordering π_1 of the vertices in layer $\mathcal{V}_1$
Output: Vertex ordering π_2 of the vertices in layer $\mathcal{V}_2$

```
1 repeat
2     for v := 1 to |V_2| − 1 do
3         if c_{v(v+1)} > c_{(v+1)v} then
4             switch vertices at positions v and v + 1;
5 until the number of crossings does not reduce;
```

3.4.2.4 Split Heuristic

The split heuristic (Algorithm 3.7), introduced by Eades and Kelly [EK86], gives better results than the median or barycenter heuristic at the expense of longer running times. The algorithm is reminiscent of *quick-sort*. First, a *pivot vertex* $v \in \mathcal{V}_2$ is chosen, and place each other vertex $u \neq v \in \mathcal{V}_2$ to the left of v if $c_{uv} < c_{vu}$, and to the right of v otherwise. After this partition, the algorithm is applied recursively to the left set and to the right set until both sets are ordered and can be concatenated. The split heuristic has a worst-case running time of $O(|\mathcal{V}_2|^2)$ but in practice it runs in time $\mathcal{O}(|\mathcal{V}_2| \log |\mathcal{V}_2|)$ if we do not consider the computation of the crossing number matrix.

Algorithm 3.7: Split Heuristic Crossing Minimization

Input : A bipartite graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ and a vertex ordering π_1 of the vertices in layer $\mathcal{V}_1$
Output: Vertex ordering π_2 of the vertices in layer $\mathcal{V}_2$

```
1 if V_2 is not empty then
2     Choose a pivot vertex v ∈ V_2;
3     V_left = ∅; V_right = ∅;
4     foreach vertex u ∈ V_2 such that u ≠ v do
5         if c_uv ≤ c_vu then
6             place u in V_left;
7         else
8             place u in V_right;
9     Recursively apply the algorithm to the graph by V_left and V_right, and output the
      concatenation of the outputs of these two applications;
```

3.4.2.5 Sifting Heuristic

The sifting heuristic was originally introduced by Rudell [Rud93] to minimize the number of vertices in *reduced ordered binary decision diagrams* problem. A reduced ordered binary decision diagrams problem is a graph which represents a boolean function and is primarily used in logic synthesis and verification. Later it was possible to adapt the sifting heuristic for the crossing minimization problem by Matuszewski, Schzönfeld, and Molitor [MSM99].

The main idea of the algorithm is to determine the optimal position for every vertex $v \in \mathcal{V}_2$ under the condition that the positions of the other vertices in layer $\mathcal{V}_2$ remain fixed. So, each vertex is placed at its locally optimal position. Since every vertex has to be set on every position, the time complexity is $O(|\mathcal{V}_2|^2)$. The sifting heuristic has a higher runtime complexity than those above; however, it produce fewer crossings in practice [Bac09]. Algorithm 3.8 presents the details of the sifting heuristic.

3.4.2.6 Two-Layer Metaheuristic Methods

Several metaheuristics were applied to solve the one-sided crossing minimization problem. a genetic algorithm to solve that problem introduced by Mäkinen [Mäk90], Mäkinen and

Sieranta [MS94] and Yamaguchi and Toh [YT00, YT01]. The results of the genetic algorithm are compared with the barycenter heuristic and induced that the genetic algorithm is better with the expense of long computation times. Similarly, a tabu search algorithm presented by Laguna, Martí, and Valls [LMV97]. The tabu search algorithm gives high-quality results but is usable only when fast computation is not necessary. The *GRASP* (greedy randomized adaptive search procedure), introduced by Laguna and Martí [LM99], gives good results especially for sparse graphs and has moderate computation times.

Algorithm 3.8: Sifting Heuristic Crossing Minimization

Input : A bipartite graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ and a vertex ordering π_1 of the vertices in layer $\mathcal{V}_1$

Output: Vertex ordering π_2 of the vertices in layer $\mathcal{V}_2$

```
1  foreach v ∈ V_2 do
2      move vertex v to the leftmost position;
3      crossings := Σ_{π2(v)<π2(u)} c_vu;
4      min_crossings := crossings;
5      for q := 1 to |V_2| − 1 do
6          crossings := crossings − c_{q(q+1)} + c_{(q+1)q};
7          switch vertices at positions q and q + 1;
8          if crossings < min_crossings then
9              min_crossings := crossings;
10             best_position := q;
11     move vertex v to position best_position;
```

3.4.2.7 Exact Two-Layer Crossing Minimization

Jünger and Mutzel [JM97] presented a branch-and-cut algorithm for solving the two-layer crossing minimization that draws on solving the *linear ordering problem*. Let δ^i_{uv} be a 0-1 variable representing the ordering of vertices i and j on layer $\mathcal{V}_i, i = 1, 2$. They present an expression for the number of crossings of a pair of permutations, π_i, which is the objective function we want to minimize.

Let us state the one-sided crossing minimization problem as an integer linear program ILP. For a two-layer hierarchical graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$, for layer $\mathcal{V}_i$ we define $\delta^i_{uv} = 1$ if $\pi_1(u) < \pi_1(v)$, otherwise $\delta^1_{uv} = 0$. For simplicity, the vertices names will be identified with their ordering index on their layer. The number of crossings could be computed with:

$$cross(\pi_2) = cross(\delta^2) = \sum_{i=1}^{|\mathcal{V}_2|-1} \sum_{j=i+1}^{|\mathcal{V}_2|} \sum_{u \in N(i)} \sum_{v \in N(j)} \delta^1_{uv} \cdot \delta^2_{ji} + \delta^1_{vu} \cdot \delta^2_{ij} \tag{3.2}$$

where the neighbours $N(v)$ of any vertex $v \in \mathcal{V}_2$ is $N(v) = \{u \in \mathcal{V}_1 : (u, v) \in \mathcal{E}\}$. The crossing number can be computed with:

$$c_{ij} = \sum_{u \in N(i)} \sum_{v \in N(j)} \delta^1_{uv}$$

Then Equation 3.2 could be represented as:

$$\begin{aligned} cross(\pi_2) &= \sum_{i=1}^{|\mathcal{V}_2|-1} \sum_{j=i+1}^{|\mathcal{V}_2|} c_{ij}\delta_{ij}^2 + c_{ji}\left(1-\delta_{ij}^2\right) \\ &= \sum_{i=1}^{|\mathcal{V}_2|-1} \sum_{j=i+1}^{|\mathcal{V}_2|} (c_{ij}-c_{ji})\,\delta_{ij}^2 \;+\; \sum_{i=1}^{|\mathcal{V}_2|-1} \sum_{j=i+1}^{|\mathcal{V}_2|} c_{ji} \end{aligned} \tag{3.3}$$

The problem of finding the vertex order with minimum number of crossings π_i could be reduced to finding the vector $\delta^i \in \{0,1\}^{\binom{|\mathcal{V}_i|}{2}}$. Assume $n_2 = |\mathcal{V}_2|$, $x_{ij} = \delta_{ij}^2$ and $a_{ij} = c_{ij} - c_{ji}$ we have to solve the linear ordering problem:

$$\min \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} a_{ij}x_{ij} \tag{3.4}$$

subject to

$$\begin{aligned} &0 \le x_{ij} + x_{jp} - x_{ip} \le 1 && \text{for} \quad 1 \le i < j < p \le n_2 \\ &0 \le x_{ij} \le 1 && \text{for} \quad 1 \le i < j \le n_2 \\ &x_{ij} \in \mathbb{Z} && \text{for} \quad 1 \le i < j \le n_2 \end{aligned}$$

The optimal crossing number can be then computed with adding $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c_{ji}$ to the minimum number of crossing computed in Equation 3.4, i.e.,:

$$cross_{opt}(\pi_2) = \min \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} a_{ij}x_{ij} + \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ji} \tag{3.5}$$

The main advantage of the ILP approach over the previous above heuristics is that it guarantees to find the optimum solution. While there is no guarantee about the termination in polynomial time, it seems to be quite successful for small to medium sized directed graphs [DETT99]. The branch-and-cut algorithm [JM97] could be used to obtain an optimal solution for directed graphs of limited size (maximum 50 vertices). According to the experiments of Bachmaier [Bac09], this approach could be applied in practice to graphs with up to 150 vertices.

3.4.3 Multi-Layer Crossing Minimization

In the *multi-layer crossing minimization problem* a k-layer hierarchical graph $\mathcal{H}$ is given, $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$ and the goal is to find permutations $\pi_1, \pi_2, \cdots, \pi_k$ such that the number of edge crossings is minimized. For the multi-layer crossing minimization problem, there are two solution approaches. In the first approach we apply a layer-by-layer one-sided two-layer heuristic recursively to get an order of the vertices in layer $\mathcal{V}_i$ while the vertices in layer $\mathcal{V}_{i-1}$ have a fixed ordering, and then proceed to the next layer. In the second approach, all the graph layers are globally considered simultaneously in the same time.

The experiments by Jünger and Mutzel [JM97] for the two-layer crossing minimization problem show that the results of the layer-by-layer sweep are far from optimum. Better results could be expected when considering all layers simultaneously, but multi-layer crossing minimization is a very hard problem [KW01, page 102]. A quick help is to start the layer-by-layer sweep several times with randomly permuted layers. This approach can tremendously improve the results in [JM97]. From the global approach side, there are some global techniques (like global sifting algorithm) presented by Bachmaier, Brandenburg, Brunner, and Hübner [BBBH11]. Experimentally, it has been shown in [BBBH11] that global sifting technique yields an improvement of 5-10% in the number of crossings over the layer-by-layer ones; however, the sifting techniques generally have a worse time complexity.

In this section, we discuss and put together different variants of the recursively layer-by-layer for the multi-layer crossing minimization beside some global techniques. We analyse the strengths and weaknesses of these algorithms both from a theoretical and computational perspective. Based on this, we present an efficient barycenter-based algorithm, we call it efficient barycenter and introduced separately in Section 3.4.3.7. A comparative study on 300 randomly generated hierarchical graphs with different number of layers, different number of vertices in each layer, and various edge densities show the efficiency of our proposed efficient barycenter algorithm.

3.4.3.1 Tutte's Algorithm

Tutte's algorithm [ES91] is considered as the first attempt for a global approach, but it does not address crossings directly. First, the x-coordinates of the vertices in the boundary (first and last) layers are fixed. In each other layer the x-coordinate of a vertex $v \in \mathcal{V}_i$, $2 \leq i \leq k-1$, is chosen as a weighted average of the x-coordinates of its neighbours in the two layers $\mathcal{V}_{i-1}$ and $\mathcal{V}_{i+1}$ according to the following equation:

$$x(v) = \frac{1}{2\ d^-(v)} \sum_{u \in N^-(v)} x(u) + \frac{1}{2\ d^+(v)} \sum_{w \in N^+(v)} x(w) \tag{3.6}$$

where, for vertex $v \in \mathcal{V}_i$, $d^-(v) = |N^-(v)|$, and $d^+(v) = |N^+(v)|$.

Now we have to solve a system of sparse linear equations to compute the value $x(v)$ for each vertex v. In the last step the vertices of each layer are sorted by their x-coordinates. The results of Tutte's algorithm are similar to one-sided two-layer barycenter heuristic [KW01].

3.4.3.2 Degree-Weighted Barycenter Algorithm (DWB)

Eades, Lin, and Tamassia [ELT96] introduced the Degree-Weighted Barycenter algorithm (DWB). In DWB algorithm, for a proper boundary s-t graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$, a boundary drawing β is chosen. This means that the x-coordinate $x(u)$ for any vertex $u \in \mathcal{V}_1 \cup \mathcal{V}_k$ belonging to the two boundary layers $\mathcal{V}_1$ and $\mathcal{V}_k$ is kept fixed. Then the x-coordinate $x(v)$ for each nonboundary vertex $v \in \mathcal{V} \setminus (\mathcal{V}_1 \cup \mathcal{V}_k)$ is computed as a kind of weighted barycenter of its neighbours in the two layers $\mathcal{V}_{i-1}$ and $\mathcal{V}_{i+1}$ as computed in Equation 3.6.

The difference between Tutte's algorithm and DWB algorithm is that the sorting step is not executed in the DWB algorithm. Thus, one round DWB algorithm has time complexity $\mathcal{O}(|\mathcal{E}|)$ since computing the x-coordinates is proportional to the degree of vertices and consequently the number of edges $|\mathcal{E}|$.

Algorithm 3.9: Degree-Weighted Barycenter DWB Crossing Minimization

Input : A proper k-layer s-t hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{E})$ with a boundary drawing $\beta(\mathcal{G})$

Output: Vertex ordering π_i of the vertices in the non-boundary layers $\mathcal{V}_2 \cup \mathcal{V}_3 \cup \cdots \cup \mathcal{V}_{k-1}$

1 Choose initial x-coordinate $x(v)$ for each nonboundary vertex $v \in \mathcal{V} \setminus (\mathcal{V}_1 \cup \mathcal{V}_k)$ in a proper boundary s-t hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$;
2 **repeat**
3 **for** $i = 2$ *to* $k-1$ **do**
4 **foreach** vertex $v \in \mathcal{V}_i$ **do**
5 $x(v) := \frac{1}{2d^-(v)} \sum_{(u,v)} x(u) + \frac{1}{2d^+(v)} \sum_{(v,w)} x(w)$;
6 **until** $x(v)$ *converges*;

3.4.3.3 Barycenter Algorithm

Barycenter algorithm [ELT96] behaves as DWB; however, it uses the barycenter values $bary(v)$ for every vertex $v \in \mathcal{V}$ computed according to Equation 3.7.

$$bary(v) = \frac{1}{2\ deg^-(v)} \sum_{u \in N^-(v)} x(u) + \frac{1}{2\ deg^+(v)} \sum_{w \in N^+(v)} x(w) \tag{3.7}$$

Then the x-coordinates of the vertices are computed after sorting the set of vertices by their barycenter values. After finishing computing the barycenter values for all vertices in all layers, then the x-coordinate of every vertex is considered as its index in its layer.

Since the vertices in layer $\mathcal{V}_i$ are sorted according to their barycenter values, the sorting step requires $\mathcal{O}(|\mathcal{V}_i| \log |\mathcal{V}_i|)$ running time. Hence, the total running time of one round for the barycenter algorithm is $\mathcal{O}(|\mathcal{E}| + \sum_{i=1}^{k} |\mathcal{V}_i| \log |\mathcal{V}_i|)$ which could be represented as $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$. Practically, the Barycenter algorithm presents results that are far from the optimum and also, it is not guaranteed that the algorithm always converges and may go into an oscillation behaviour.

3.4.3.4 Global Sifting Algorithm

Based on the sifting heuristic [MSM99] (introduced in Section 3.4.2), Bachmaier, Brandenburg, Brunner, and Hübner [BBBH11] have introduced the global sifting algorithm. In the

Algorithm 3.10: Barycenter Algorithm Crossing Minimization

Input : A proper k-layer s-t hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{E})$
Output: Vertex ordering π_i of the vertices in all the graph layers $\mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_k$ including the boundary layers

Choose initial x-coordinate $x(v)$ for each vertex $v \in \mathcal{V}$ in $\mathcal{H}$;
repeat
  for $i = 1$ *to* k **do**
    foreach vertex $v \in \mathcal{V}_i$ **do**
      $bary(v) := \frac{1}{2d^-(v)} \sum_{(u,v)} x(u) + \frac{1}{2d^+(v)} \sum_{(v,w)} x(w)$;
  Sort the vertices of layer $\mathcal{V}_i$ into increasing order of $bary(v)$;
  Let the x-coordinate of vertices in $\mathcal{V}_i$ be their sorted indices;
until $x(v)$ *converges*;

global sifting algorithm, for a proper hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$, a block $\mathcal{A}$ is built containing either all dummy vertices of each long edge or an original vertex, afterwards a list of blocks $\mathcal{B}$ is created. Then it finds (in one step) the best position for the entire block in the list of blocks. All the vertices of a block get the same x-coordinate and, thus, the ordering causes no crossings produced by edges connecting dummy vertices, i.e. it guarantees the absence of type-2 conflict crossings.

Algorithm 3.11: Global Sifting Crossing Minimization

Input : Proper k-layers hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$ and number ρ of sifting rounds
Output: a hierarchical graph $\mathcal{H}$ in which the vertices are ordered by value $\pi(v)$ for each $v \in \mathcal{V}$

Create list $\mathcal{B}$ of all blocks in proper hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$;
for $1 \leq i \leq \rho$ **do**
  foreach block $\mathcal{A} \in \mathcal{B}$ **do**
    $\mathcal{A} \leftarrow$ SIFTING_STEP $(\mathcal{G}, \mathcal{B}, \mathcal{A})$
foreach $v \in \mathcal{V}$ **do**
  $\pi(v) \leftarrow \pi(block(v))$

As an initialization, the list of blocks $\mathcal{B}$ is sorted arbitrarily and each block $\mathcal{A}$ gets its block order $\pi(\mathcal{A})$ in $\mathcal{B}$ as its position in $\mathcal{B}$. At any time during the execution of the algorithm, interpreting $\pi(\mathcal{A})$ for each block $\mathcal{A}$ as an x-coordinate for each vertex v in block A and $\phi(v)$, which is the number of the layer of v, as its y-coordinate results in a drawing respecting the current ordering of $\mathcal{B}$. There, all positions for a block A are tested and A is moved to that position where it has the fewest crossings. This is done for each block $A \in \mathcal{B}$ in the shifting step and repeated a certain number of times ρ. Finally, each vertex is set to the position of its block.

Based on the global sifting algorithm, the global barycenter and the global median algorithms are also proposed [BBBH11], where we iteratively take the $\pi-$positions of the blocks in the blocks list $\mathcal{B}$ and compute the *barycenter* or *median* of each block, respectively, and then

sort $\mathcal{B}$ according to these values. A recent experimental study [BBBH11] has shown that the global sifting algorithm is computationally the most efficient method to minimize the edge crossing. However, global sifting is also the slowest of the above algorithms and has a running time $\mathcal{O}\left(|\mathcal{E}|^2\right)$ which is quadratic in the number of edges of the input graph.

3.4.3.5 Exact Multi-Layer Crossing Minimization

Jünger, Lee, Mutzel, and Odenthal [JLMO97] introduced an ILP model for computing the minimum crossing number for multi-layer hierarchical graphs presented in Equation 3.8. To simplify the description, the vertex names will be identified with their ordering number on their layers. For levels $1 < i \leq k$ and all pairs of edges $(u,v),(x,y) \in \mathcal{E}_i$, where $\mathcal{E}_i$ is the set of short edges connecting the vertices in the two consecutive layers $\mathcal{V}_{i-1}$ and $\mathcal{V}_i$, we define $c^i_{uvxy} = 1$ if (u,v) and (x,y) cross and $c^i_{uvxy} = 0$, otherwise. The variables x^i_{ux} in Equation 3.8 store after solving it the desired relative vertex positions on layer $1 \leq i \leq k$. A value of $x^i_{ux} = 1$ means $u \prec x$ and a value of $x^i_{ux} = 0$ means $u \succ x$.

Jünger Lee, Mutzel, and Odenthal [JLMO97] stated that their branch-and-cut approach is only practicable if additional and deeper polyhedral studies are conducted for speed up. According to the implementation within the Gravisto framework [BBFMM04], the approach can be applied to graphs with up to 40 vertices in practice [Bac09].

$$cross_{opt} = \min \sum_{i=2}^{k} \sum_{\substack{(u,v),(w,y)\in\mathcal{E}_i,\\ 1\leq u<w\leq|\mathcal{V}_i|,\\ 1\leq v\neq y\leq|\mathcal{V}_i|}} c^i_{uvwy} \tag{3.8}$$

subject to

$$-c^i_{uvxy} \leq x^i_{vy} \leq x^{i-1}_{uw} \leq c^i_{uvxy} \quad \text{for} \quad \begin{array}{l}(u,v),(w,y) \in \mathcal{E}_i,\\ 1 \leq u < w \leq |\mathcal{V}_{i-1}|,\\ 1 \leq v < y \leq |\mathcal{V}_i|,\\ 1 < i \leq k\end{array}$$

$$1 - c^i_{uvxy} \leq x^i_{uv} - x^{i-1}_{uw} \leq 1 + c^i_{uvxy} \quad \text{for} \quad \begin{array}{l}(u,v),(w,y) \in \mathcal{E}_i,\\ 1 \leq u < w \leq |\mathcal{V}_{i-1}|,\\ 1 \leq y < v \leq |\mathcal{V}_i|,\\ 1 < i \leq k\end{array}$$

$$0 \leq x^i_{uw} + x^i_{wz} - x^i_{uz} \leq 1 \quad \text{for} \quad \begin{array}{l}u,w,z \in \mathcal{V}_i,\\ 1 \leq u < w < z \leq |\mathcal{V}_i|,\\ 1 \leq i \leq k\end{array}$$

$$x^i_{uw} \in \{0,1\} \quad \text{for} \quad \begin{array}{l}u,w \in \mathcal{V}_i,\\ 1 \leq u < w \leq |\mathcal{V}_i|,\\ 1 \leq i \leq k\end{array}$$

$$c^i_{uvxy} \in \{0,1\} \quad \text{for} \quad \begin{array}{l}(u,v),(w,y) \in \mathcal{E}_i,\\ 1 \leq u < w \leq |\mathcal{V}_{i-1}|,\\ 1 \leq v < y \leq |\mathcal{V}_i|,\\ 1 < i \leq k\end{array}$$

3.4.3.6 Multi-Layer Metaheuristic Methods

Many different metaheuristic techniques have been proposed in literature to be used in the global multi-layer crossing minimization problem. An evolutionary algorithm has been introduced by Utech, Branke, Schmeck, and Eades [UBSE98] and genetic algorithms have been considered by Kuntz, P. and Pinaud, B. and Lehn [KPL06]. Also, tabu search has been applied by Laguna, Martí, and Valls [LMV97], and windows optimization by Eschbach, Günther, Drechsler, and Becker [EGDB02]. These general (stochastic) global search approaches usually compute good solutions with few crossings at the expense of high running times [Bac09].

3.4.3.7 Efficient Barycenter Algorithm

It is clear that the DWB algorithm is limited by a necessity to choose a boundary drawing β, i.e. the vertices in the boundary layers $\mathcal{V}_1$ and $\mathcal{V}_k$ have fixed position in a hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$. In other words, as noted in [ELT96], the performance of DWB algorithm is poor, except for a good choice of β. The barycenter algorithm, although it does not need a boundary drawing β, in many cases we found it enters in an *oscillation* behaviour and never converges and produces not good results.

Here, we introduce a new barycenter-based algorithm and call it efficient barycenter algorithm. It combines the theoretical advantages of both the DWB algorithm and barycenter algorithm. Our proposed efficient barycenter algorithm (as we later show) gives computationally better (similar) results than the global sifting algorithm. Like the barycenter algorithm, it is not restricted by a choice of β. Moreover, like the DWB algorithm, it computes the x-coordinates $x(v)$ for every vertex $v \in \mathcal{V}_i$ in a steady-state way. The *updated* x-coordinates for adjacent vertices u in the previous layer $\mathcal{V}_{i-1}$ are used for this. The idea is to go directly to the sorting step after finishing computing the barycenter values for all vertices $u \in \mathcal{V}_{i-1}$, in order to compute the x-coordinates $x(u)$ for these vertices before starting to compute the barycenter values for the vertices in layer $\mathcal{V}_i$. Detailed steps of the efficient barycenter algorithm are given in Algorithm 3.12.

The efficient barycenter algorithm works with *any* proper hierarchical graph, not only with *boundary* s-t graphs as the DWB algorithm or s-t graphs as the barycenter algorithm. If any vertex v belongs to an intermediate layer $\mathcal{V} \setminus \mathcal{V}_1 \cup \mathcal{V}_k$, i.e. $v \in \mathcal{V}_i, 2 \leq i \leq k-1$, has either no outgoing edges, i.e. $N^+(v) = \emptyset$ or $d^+(v) = 0$, or no incoming edges, i.e. $N^-(v) = \emptyset$ or $d^-(v) = 0$, it is easily taken into account by the updated form of the barycenter equation used in these algorithms (Equation 3.9). This also avoids introducing dummy vertices to convert an arbitrary hierarchical graph to an s-t graph, a technique common to existing algorithms [EFN06].

$$bary(v) = \begin{cases} \dfrac{1}{d^-(v)} \displaystyle\sum_{u \in N^-(v)} x(u) & \text{if } d^+(v) = 0; \\ \dfrac{1}{d^+(v)} \displaystyle\sum_{w \in N^+(v)} x(w) & \text{if } d^-(v) = 0; \\ \dfrac{1}{2\ d^-(v)} \displaystyle\sum_{u \in N^-(v)} x(u) + \dfrac{1}{2\ d^+(v)} \displaystyle\sum_{w \in N^+(v)} x(w) & \text{otherwise;} \end{cases} \tag{3.9}$$

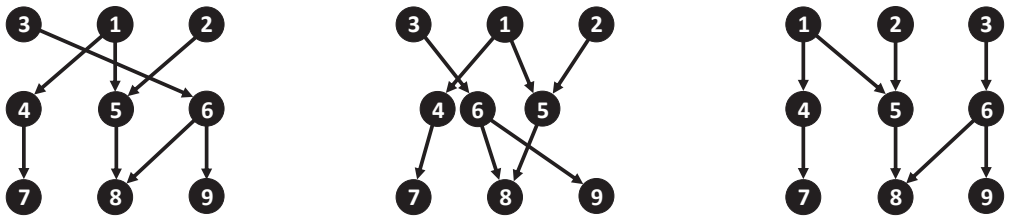
Algorithm 3.12: Efficient Barycenter Crossing Minimization

Input : A proper k-layers (not necessarily s-t) hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$

Output: A hierarchical graph $\mathcal{H}$ with vertices ordered by value $bary(v)$ for each vertex $v \in \mathcal{V}_i, 1 \leq 1 \leq k$

```
1  Choose initial x-coordinate x(v) for each vertex v in a not restricted s-t hierarchical
   graph H = (V_1, V_2, ..., V_k; E);
2  repeat
3      for i = 1 to k do
4          foreach vertex v ∈ V_i do
5              if d^-(v) = 0 then
6                  bary(v) ← 1/d^+(v) Σ_(v,w) x(w);
7              else if d^+(v) = 0 then
8                  bary(v) ← 1/d^-(v) Σ_(u,v) x(u);
9              else
10                 bary(v) ← 1/(2d^-(v)) Σ_(u,v) x(u) + 1/(2d^+(v)) Σ_(v,w) x(w);
11         Sort the vertices of layer V_i into increasing order of b;
12         Let the x-coordinate of vertices in V_i be their sorted indices;
13 until x(v) converges;
```

Although, the DWB algorithm has a better time complexity than the efficient barycenter algorithm, it is limited by the choice of the boundary drawing β, see the example given in Figure 3.6. Keeping a boundary drawing β of the drawing in Figure 3.6(a), the DWB algorithm gives a *non-planar* drawing in Figure 3.6(b), where the efficient barycenter algorithm produces the *planar* drawing in Figure 3.6(c). The graph is planar and a planar drawing is obtained by efficient barycenter algorithm. However, the graph is *not* β-planar and the DWB algorithm algorithm *cannot* produce a planar drawing of the graph.



(a) (b) (c)

Figure 3.6: An example of the effect of a good choice of boundary drawing β. An initial drawing (a), a non-planar drawing (b) produced by the DWB algorithm, and a planar drawing (c) produced by the efficient barycenter algorithm.

3.4.3.8 Computational Results

We have implemented the median algorithm and the efficient median algorithm in which the barycenter values are computed in a similar way as in the barycenter algorithm and the efficient barycenter algorithm respectively. In the median algorithm and efficient median algorithm, the barycenter value of every vertex $v \in \mathcal{V}_i$ is computed as the average of the two x-coordinates of its two medians in the two layers $\mathcal{V}_{i-1}$ and $\mathcal{V}_{i+1}$.

In this section we provide the results of a detailed comparative study. We compare the following algorithms: the one-sided 2-layer barycenter heuristic (BH), the one-sided 2-layers median heuristic (MH), the degree-weighted barycenter (DWB), the barycenter algorithm (BC), the median algorithm (MD), the global sifting (GS), the global barycenter (GB), the global median (GM), the efficient median algorithm (FMD), and the efficient barycenter algorithm (FBC). We deal with both sparse and dense hierarchical graphs and also small and large graphs. The implementations of the three algorithms global sifting, global barycenter, and global median implemented are done through the Gravisto graph drawing tool [BBFMM04].

Sparse graphs are generated as $k \times b$ hierarchical graphs, i.e., k-layer hierarchical graphs $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$ with $|\mathcal{V}_i| = b$, $1 \leq i \leq k$. Each vertex $v \in \mathcal{V}_i$, $1 \leq i \leq k-1$, has two outgoing edges connecting it to two randomly chosen vertices in layer $\mathcal{V}_{i+1}$. Hence, the total number of edges will be $|\mathcal{E}| = 2 \cdot b \cdot (k-1)$. Two types of sparse graphs are considered; in the first type, the number of layers is kept fixed to $k = 5$ where the number of vertices in each layer $|\mathcal{V}_i|$ varies as $|\mathcal{V}_i| = 10, 20, \cdots, 100$, and 10 random graph instances have been generated for each number of vertices $|\mathcal{V}_i|$. Results for this type of sparse hierarchical graphs for the number of crossing and execution time are presented in Figure 3.7.

In the second type, the number of vertices in each layer is kept fixed to $|\mathcal{V}_i| = 10$ vertices where the number of layers k vary as $k = 3, 4, \cdots, 12$, and generating 10 random instances for each number of layers k. Results for this type of sparse hierarchical graphs for the number of crossing and execution time are presented in Figure 3.8.

For generating dense hierarchical graphs, we consider 10 different edge densities D which vary as $D = 0.20, 0.25, \cdots, 0.65$. The number of layers is kept fixed to $k = 5$ and the number of vertices in each layer is kept fixed to $|\mathcal{V}_i| = 10$. The number of edges that the dense graph must contain is computed by multiplying the maximum number of edges the graph could have $\left(2 \times (k-1) \times |\mathcal{V}_i|^2\right)$ by the desired density D. An edge is added randomly by choosing two vertices in two consecutive layers. The process of adding the remaining edges is repeated until all the edges are added such that every vertex $v \in \mathcal{V}_i$, $1 \leq i \leq k$ has at least two outgoing edges. 10 random samples are generated for each edge density. The results for dense hierarchical graphs for the number of crossing and execution time are presented in Figure 3.9.

Furthermore, we have dealt with very large graphs. The considered large graphs are equipped with number of vertices from $|\mathcal{V}| = 1000$ till $|\mathcal{V}| = 10000$ vertices in the graph. We generate large graphs as sparse ones (each vertex having two outgoing edges) keeping the number of vertices in each layer fixed to $|\mathcal{V}_i| = 100$ vertex where the number of layers k varies as $k = 10, 20, \cdots, 100$. 10 random instances for each number of layers k have been randomly generated. Results for this type of sparse hierarchical graphs for the number of crossings and execution time are presented in Figure 3.10.

In the four Figures 3.7-3.10, the charts in subfigures (a) show the results of the number of crossings of the considered algorithms, where the two charts in the two subfigures (b) and (c) in each figure present results of the running time. The charts in the subfigures (b) show the results of the running time of all the considered algorithms, where the charts in the subfigures (c) present results of the running time in a time interval in which the efficient barycenter algorithm can be easily compared against the other algorithms.

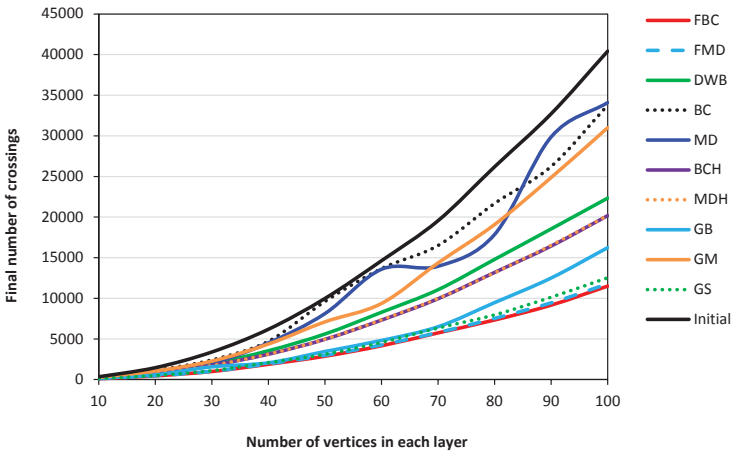
The following remarks summarize the major results of the experimental study:

- In all the cases (with different number of vertices, number of layers and edge densities) the efficient barycenter algorithm produces crossings about 17% *lower* than the barycenter heuristic, the median heuristics, the global barycenter, and the global median, 43% *lower* than the degree-weighted barycenter, 22% *lower* than the barycenter algorithm, and 4% *lower* than the global sifting.
- For small (lower density) graphs, the efficient barycenter produces the *lowest* number of crossings (Figure 3.11 gives an example).
- The barycenter algorithm and the median algorithm could not achieve convergence (at least in 100 iterations for small graphs and 1000 iteration for large ones) with most of the samples and go to oscillation behaviour.
- In some cases, the minimum number of edge crossings is obtained before the convergence for most of the considered algorithms. Hence, care must be taken to ensure so-called *elitism* in various heuristics. Hence, we recommend that the drawing with minimum number of crossings should be selected from the initial drawing and all the drawings produced iteratively till the convergence.
- There are slightly better results of the barycenter algorithm and the efficient barycenter against the median algorithm and efficient median algorithm, respectively. Also the median algorithm could not achieve convergence with many instances and goes into an oscillation behaviour, as barycenter algorithm, specially with dense and large graphs.

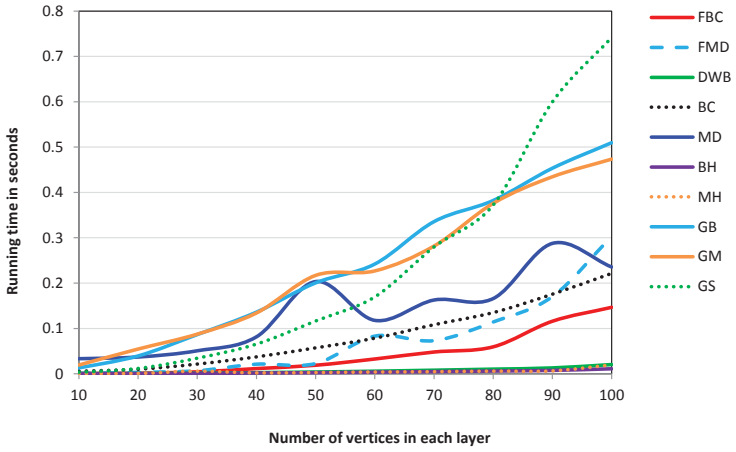
3.5 Coordinate Assignment

In the last phase of the Sugiyama approach, *coordinate assignment*, we compute an x-coordinate $x(v)$ for every vertex $v \in \mathcal{V}$. It is usually constrained to preserve the ordering determined in the third phase and to introduce a minimum separation space δ between vertices within a layer. The y-coordinates are considered as the layers numbers. Finally, the dummy vertices introduced to break long edges have no geometric representation, i.e. they are removed and replaced by edge bends. During the coordinates assignment, there are two main objectives. The first is that the drawing should have as few edge bends as possible, and the second is that the drawing should be as compact as possible.

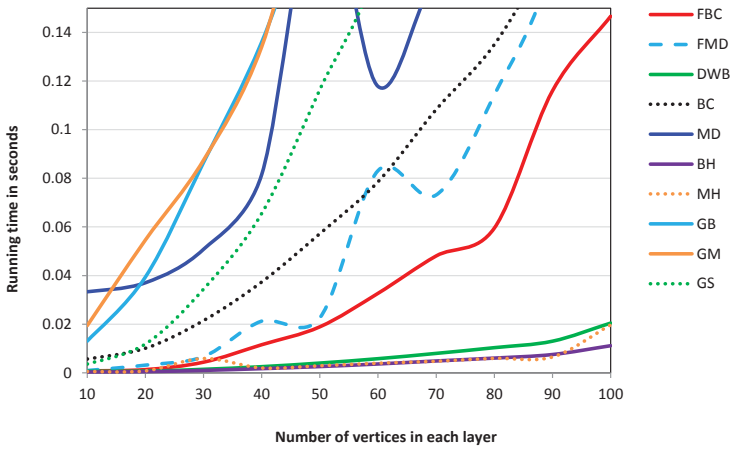
Actually, there are several techniques for the horizontal coordinates assignment [STT81, GNV88, ES91, ELT96, San96, San99, BJL01, BK02, Ism04]. Generally, these techniques use different approaches of various optimization models and also different iterative techniques. We will present some exact algorithms for the horizontal coordinates assignment and afterwards a heuristic is introduced. In the following techniques, it is assumed that the input graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$ is a proper k-layer hierarchical graph.



(a)

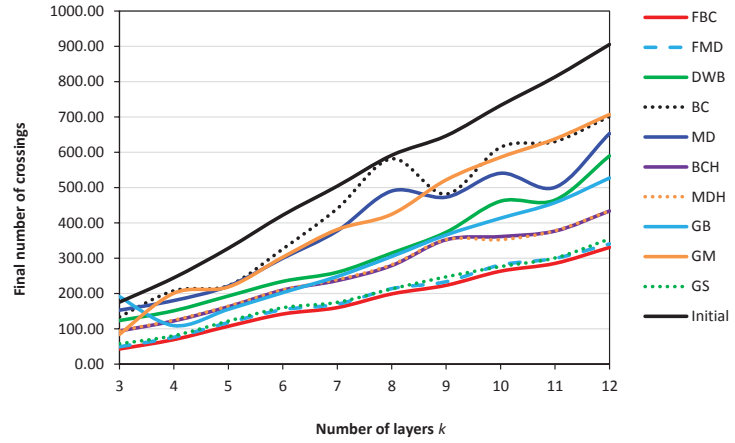


(b)

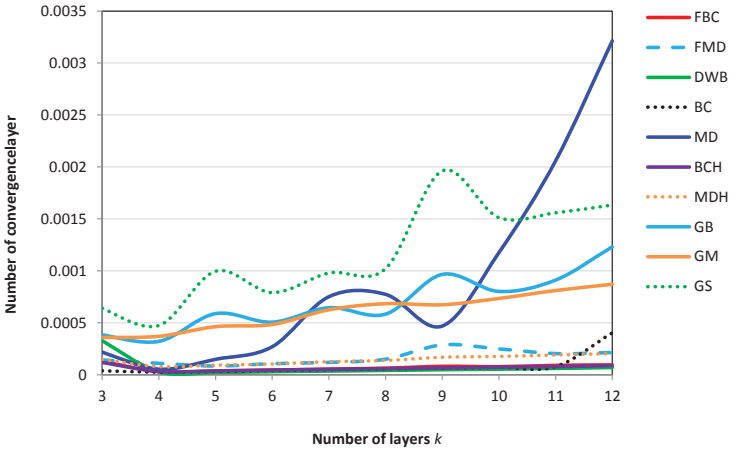


(c)

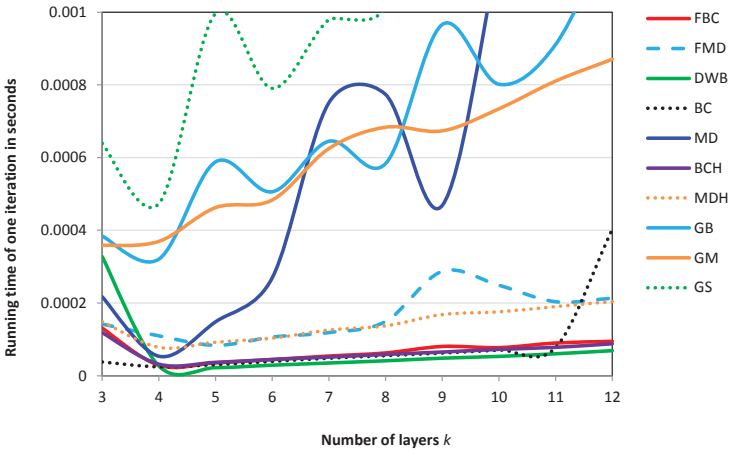
Figure 3.7: Results for sparse graphs with fixed number of layers $k = 5$ and different number of vertices in each layer $|\mathcal{V}_i| = 10, 20, ..., 100$.



(a)

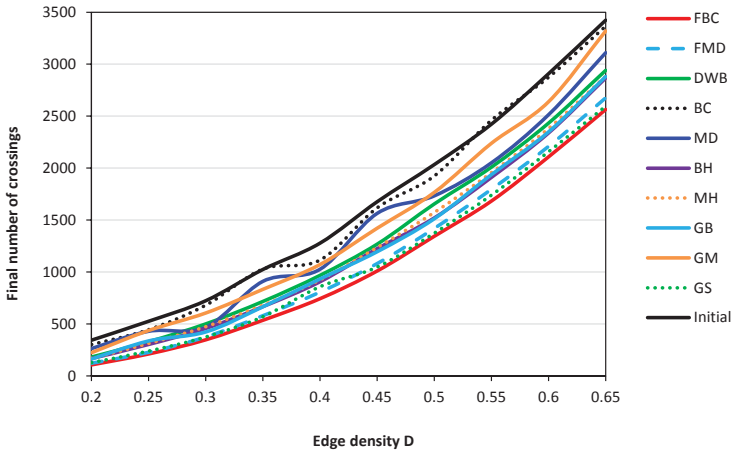


(b)

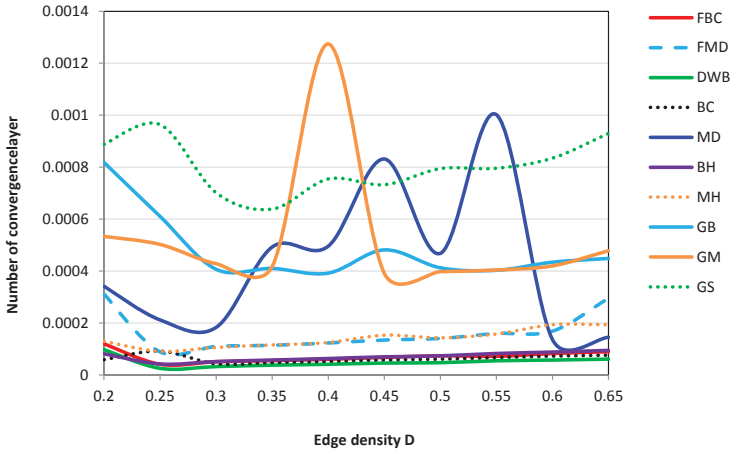


(c)

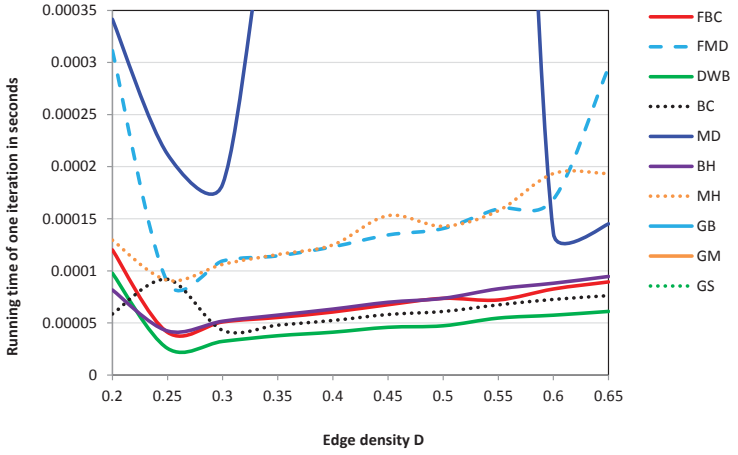
Figure 3.8: Results for sparse graphs with fixed number of vertices in each layer $|\mathcal{V}_i| = 10$ and different number of layers $k = 3, 4, \cdots, 12$.



(a)

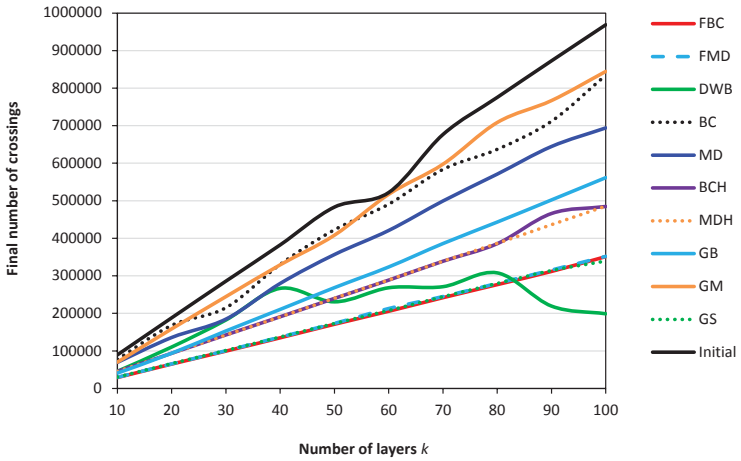


(b)

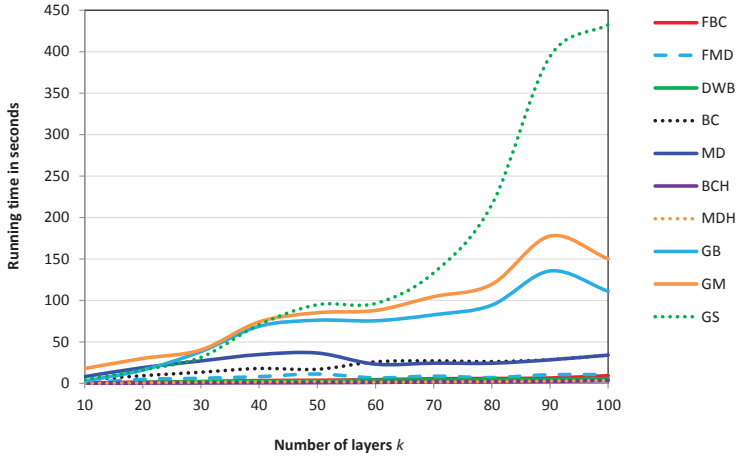


(c)

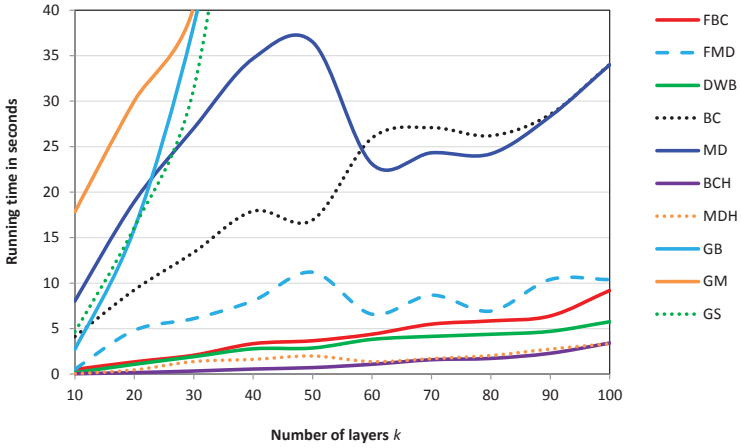
Figure 3.9: Results for dense graphs with different edge densities $D = 0.20, 0.25, \cdots, 0.65$, a fixed number of vertices in each layer $|\mathcal{V}_i| = 10$, and a fixed number of layers $k = 5$.



(a)

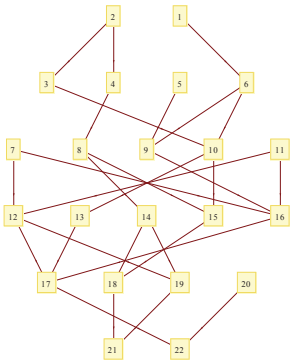


(b)

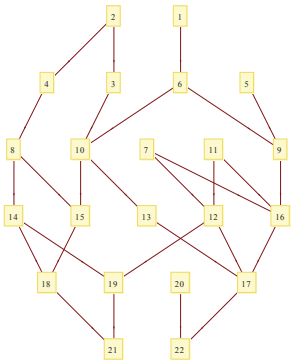


(c)

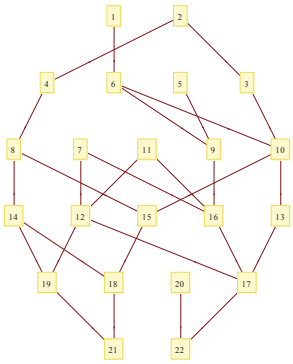
Figure 3.10: Results for large sparse graphs with fixed number of vertices in each layer $|\mathcal{V}_i| = 100$ and different number of layers $k = 10, 20, \cdots, 100$.



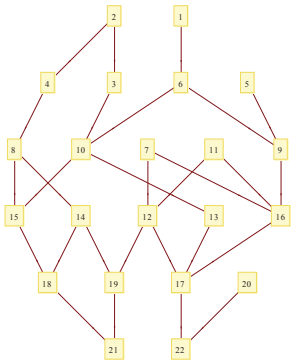
(a) Initial drawing (29 crossings)



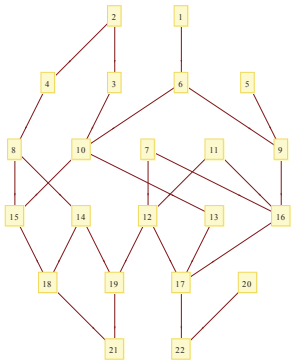
(b) Final drawing using the efficient barycenter (3 crossings)



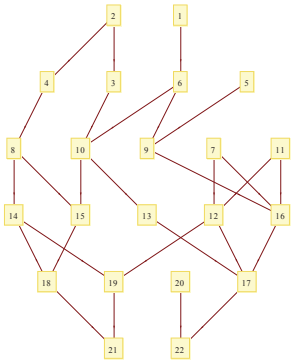
(c) Final drawing using the DWB (12 crossings)



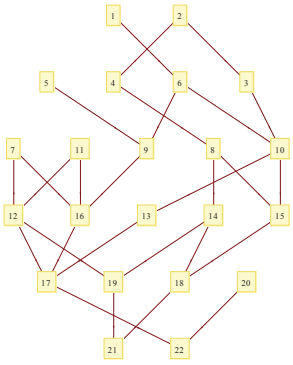
(d) Final drawing using the barycenter heuristic (4 crossings)



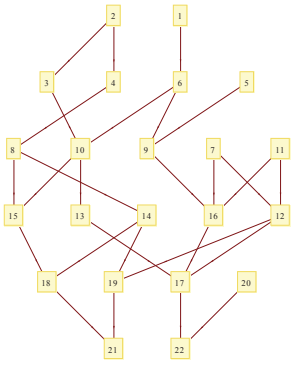
(e) Final drawing using the median heuristic (4 crossings)



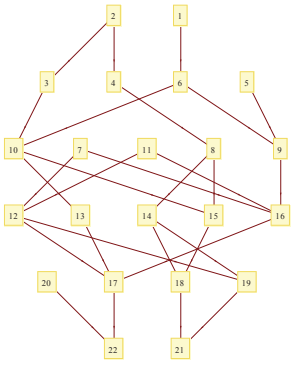
(f) Final drawing using the barycenter algorithm (8 crossings)



(g) Final drawing using the global sifting (9 crossings)



(h) Final drawing using the global barycenter (8 crossings)



(i) Final drawing using the global median (19 crossings)

Figure 3.11: Performance of various algorithms for minimizing crossing of the same drawing of a hierarchical graph.

3.5.1 Exact Algorithm

The problem of horizontal coordinates assignment could be stated as an optimization problem. Consider the directed path $p = (v_1, v_2, \cdots, v_q)$, where $v_2, v_3, \cdots, v_{q-1}$ are dummy vertices. An edge that is represented in the drawing via a directed path p is called an *edge-path*. In order to draw the path p as a straight line, then for $2 \leq i \leq q$, the x-coordinate $x(v_i)$ of the dummy vertex v_q should be computed as:

$$x(v_i) - x(v_1) = \frac{i-1}{q-1}\left(x(v_q) - x(v_1)\right) \tag{3.10}$$

Note that this formula is only valid for equidistant layers, where it is straightforward to update it for unequal layer separation distances.

In order to get a compact representation of the formula in Equation 3.10, we define $\bar{x}(v_i) = x(v_1) + \frac{i-1}{q-1}\left(x(v_q) - x(v_1)\right)$ which will be the x-coordinate of vertex v_i if v_i lie on the straight line that connects between $x(v_1)$ and $x(v_q)$. Now, the deviation of the path from a straight line can be formulated as:

$$\sum_{i=2}^{q-1} \left(x(v_i) - \bar{x}(v_i)\right)^2$$

To draw the edge as straight as possible, the summation of the deviation should be minimized, i.e.,

$$\min \sum_{p(e) \text{ with } e \in \mathcal{E}} \sum_{i=2}^{q-1} \left(x(v_i) - \bar{x}(v_i)\right)^2 \tag{3.11}$$

subject to

$$x(w) - x(v) \geq \rho(w, v) \quad \text{for} \quad \phi(w) = \phi(v), w \succ v$$

Since the model in Equation 3.11 is a quadratic function, it can be solved for small instances only. Every not necessarily proper level planar graph has a planar straight-line drawing which can be computed in $\mathcal{O}(|\mathcal{V}|^2)$ time [EFN06].

Another linear objective to draw the lines as close to vertical lines as possible is introduced by Gansner, Koutsofios, North, and Vo [GKNV93]. This objective is presented as follows:

$$\min \sum_{(u,v) \in \mathcal{E}} \Omega\left((u, v)\right) \cdot |x(u) - x(v)| \tag{3.12}$$

subject to

$$x(w) - x(v) \geq \rho(w, v) \quad \text{for} \quad \phi(w) = \phi(v), w \succ v$$

where $\Omega(e)$ denotes the priority to draw edge $e \in \mathcal{E}$ vertically. Many authors suggest that the higher priorities will be $\Omega(e) = 8$ for inner segments, $\Omega(e) = 2$ for outer segments incident to exactly one dummy vertex, and $\Omega(e) = 1$ for all other outer segments. The linear program can be interpreted as a *rank assignment problem* on a compaction hierarchical graph $\mathcal{H}_a = (\mathcal{V}, \{(u, v) : u, v \in \mathcal{V}_i, |\pi(u) - \pi(v)| = 1, 1 \leq i \leq k\}$ with length function δ. Each valid rank assignment corresponds to a valid drawing. The objective function can be modelled by adding vertices and edges to the graph $\mathcal{H}_a$ [GKNV93, KW01].

The main drawback of the vertical edges approach is, that an edge can have a number of bends equal to its number of dummy vertices, which consequently reduces the drawing readability. To avoid this negative behaviour, the linear segments model was proposed, where each edge is drawn as polyline with at most three segments and two bends, such that the middle segment is always drawn vertically. In general, linear segment drawings have fewer bends but normally need more width area. Some algorithms have been proposed for this model [BK02, BJL01, San95], where the technique produced by Brandes and Köpf [BK02] produces good results in linear time.

3.5.2 Heuristic Technique

Another possibility to obtain the horizontal coordinates assignment is to use an improvement heuristic. Generally, any heuristic for this can roughly be considered similar to the following algorithm framework: The *initial coordinates* for the vertices could be considered with

```
1 initial coordinates;
2 while some condition do
3 |   positioning;
4 |   straightening;
5 |   packing;
```

minimal distance from left to right in the order given by the crossing minimization. In the *positioning* step, the ideas applied in the crossing minimization section, like median or barycenter, between two layers might be considered. Another idea is to think of the vertices as balls and the edges as strings of a pendulum [San99]. In order to minimize the edge bends produced from the last two steps, in the *straightening* step one could try to assign the dummy vertices of an edge-path to the same x-coordinate. Hence, the edges can be seen as rubber bends with vertices at both ends and the dummy vertices in between. This increases the width and consequently enlarges the drawing in x-direction. Finally, the drawing is compressed in the *packing* step by moving the vertices closer together again without introducing new bends. These steps might be iterated to obtain a satisfying drawing.

3.6 Combining Steps

Normally, the single steps of the Sugiyama approach are performed independently. Even, some steps specially the layering step and crossing minimization step are strongly dependant and it seems to be reasonable to solve them together in one step. A convincing example is depicted in Figure 3.12. A first attempt using an evolutionary algorithm is presented by Utech, Branke, Schmeck, and Eades et al. [UBSE98]. Recently, Bachmaier, Brunner, and Gleißner [BBG11] introduced the grid sifting algorithm which combines the layering and crossing minimization steps together. The grid sifting algorithm is based on the sifting technique, which prioritizes few crossings over few levels. Another recent algorithm, called upward planar representation (UPR), presented by Chimani, Gutwenger, Mutzel, and Wong [CGMW11], which computes the layering of the vertices in order to minimize edge crossings. The UPR algorithm greatly improves the drawing quality and leads to good hierarchical drawings with few crossings.

(a) (b)

Figure 3.12: Example of combining the Sugiyama approach. A directed acyclic graph with 4 vertices and 4 edges which be hierarchically drawn either in two layers but with crossings (as in drawing (a)) or in at least three layers without crossings but with long edges (as in drawing (b)).

3.7 Generating Random Hierarchical Graphs

There are four possibilities to deal with random hierarchical graphs:

1. Generating an undirected graph, giving directions to the edges, then removing the cycles from the graph, and finally layering the vertices into layers.
2. Generating a directed graph, removing the cycles from the graph, and then layering the vertices into layers.
3. Generating a directed acyclic graph and then layering the vertices into layers.
4. Generating a layered hierarchical graph directly.

The general way of generating random hierarchical graphs is to generate a directed acyclic graph and then transfer the graph to a layering technique (such as the Coffman-Graham algorithm) in order to get the layered graph of the initial directed acyclic graph. This way has been used quite often [BBBH11, CGMW10].

So, in this section we try to get an answer to the following questions: *what is the better way for generating a hierarchical graph randomly: generating a directed acyclic graph and then layering it or generating the hierarchical graph directly?*

During our experiments in the previous section, we observed that the edge density of the hierarchical graph changes in both cases, keeping the number of vertices and edges the same in both graphs. The graph edge density is the ratio of the actual number of edges the graph has to the maximum number of edges the graph could have (when it is a complete graph and there is an edge between every pair of vertices). Till now, there is formula for the upper bound of number of edges in a hierarchical.

Assume that $\mathcal{G} = (\mathcal{V}, \mathcal{E}_{dag})$ is a *directed acyclic graph* (dag) with $n = |\mathcal{V}|$ vertices and $m_{dag} = |\mathcal{E}_{dag}|$. The maximum number of edges $m_{dag-max}$ is:

$$m_{dag-max} = \frac{n\,(n-1)}{2} \tag{3.13}$$

and hence, the edges density D_{dag} is:

$$D_{dag} = \frac{m_{dag}}{m_{dag-max}} = \frac{m_{dag}}{\left(\frac{n(n-1)}{2}\right)} = \frac{2 \cdot m_{dag}}{n\,(n-1)} \tag{3.14}$$

Assume $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k; \mathcal{E})$ is the hierarchical graph of the directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}_{dag})$, such that $\mathcal{V} = \mathcal{V}_1 \cup V_2 \cup \cdots \cup \mathcal{V}_k$, $n_i = |\mathcal{V}_i|$, $1 \leq i \leq k$ and $m = |\mathcal{E}|$. According to the hierarchical graph properties, edge $(u, v) \in \mathcal{E}$ where $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ this implies $j \neq i$ and $j \geq i$. This means that there is no edge that could connect two vertices belonging to the same layer, but this edge set are permitted in the directed acyclic graph before implementing the layering step. We call this set of *forbidden edges* $\mathcal{E}^*$ and the number of these forbidden edges is $m^* = |\mathcal{E}^*|$. So, there is a set of edges in a directed acyclic graph where these edges are forbidden in a hierarchical graph, and hence:

$$\mathcal{E} = \mathcal{E}_{dag} \setminus \mathcal{E}^* \quad \text{and} \quad m = m_{dag} - m^*$$

The number of the forbidden edges m_i^* between the set of vertices in layer $\mathcal{V}_i$ in $\mathcal{H}$ is:

$$m_i^* = 1 + 2 + \cdots + (n_i - 1) = \frac{n_i\,(n_i - 1)}{2} \tag{3.15}$$

and hence, the total number of the forbidden edges m^* between all the vertices $\mathcal{V}$ in $\mathcal{H}$ is:

$$m^* = \sum_{i=1}^{k} m_i^* = \sum_{i=1}^{k} \frac{n_i\,(n_i - 1)}{2} = \sum_{i=1}^{k} \frac{n_i^2}{2} - \sum_{i=1}^{k} \frac{n_i}{2} = \frac{1}{2}\left[\left(\sum_{i=1}^{k} n_i^2\right) - n\right] \tag{3.16}$$

Now, for a hierarchical graph $\mathcal{H}$, the maximum number of permitted edges m_{max} could be computed as:

$$m_{max} = m_{dag-max} - m^* = \frac{n\ (n-1)}{2} - \frac{1}{2}\left[\left(\sum_{i=1}^{k} n_i^2\right) - n\right] = \frac{1}{2}\left(n^2 - \sum_{i=1}^{k} n_i^2\right) \tag{3.17}$$

Hence, the edge density D for a general not necessarily proper hierarchical graph H is:

$$D = \frac{m}{m_h^*} = \frac{m}{\frac{1}{2}\left(n^2 - \sum_{i=1}^{k} n_i^2\right)} = \frac{2 \cdot m}{n^2 - \sum_{i=1}^{k} n_i^2} \tag{3.18}$$

A proper hierarchical graph $\mathcal{H}_p = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k; \mathcal{E}_p)$ with k layers, $m_p = |\mathcal{E}_p|$, is considered as a sequence of $k-1$ consecutive bipartite graphs $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{V}_{i+1}; \mathcal{E}_i)$, $1 \leq i \leq k-1$, i.e. $\mathcal{H}_p = \bigcup_{i=1}^{k-1} \mathcal{G}_i$, and $\mathcal{E}_p = \bigcup_{i=1}^{k-1} \mathcal{E}_i$. Any edge in proper hierarchical graph $\mathcal{H}_p$ connects vertices only belonging to consecutive layers, i.e., if $(u, v) \in \mathcal{H}_p$ then $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_{i+1}$, $1 \leq i \leq k-1$. Hence, the maximum number of edges $m_{\mathcal{H}_p}^*$ in a proper hierarchical graph $\mathcal{H}_p$ is:

$$m_p^* = \sum_{i=1}^{k-1} (n_i \times n_{i+1})$$

And then, the edge density D_p for a proper hierarchical graph is:

$$D_p = \frac{m_p}{m_p^*} = \frac{m_p}{\sum_{i=1}^{k-1} (n_i \times n_{i+1})} \tag{3.19}$$

Here we introduce two algorithms (algorithms 3.13,3.14) for generating proper and general hierarchical graphs. In these two generators, the following symbols will be used:

$\underline{k}$ as the minimum number of layers,

$\bar{k}$ as the maximum number of layers,

$\underline{n}$ as the minimum number of vertices in a layer,

$\bar{n}$ as the maximum number of vertices in a layer,

d^+ as the minimum outdegree of a vertex,

D as the graph edge density, and

l^* as the ration of the long edges in the graph.

3.8 Summary

In this chapter, we present the hierarchical approach for creating polyline (or straight-line) drawings of directed graphs with vertices arranged in horizontal layers. This approach was presented in 1981 by Sugiyama, Tagawa and Toda [STT81], and several subsequent methods [GKNV93, Car80, ES91, GNV88, GM89, Mes88, MRH91, PT90] are actually closely related. These methods are highly intuitive and can be applied to any digraph, regardless of its different theoretical and topological characteristics. The four main steps of the Sygiyama approach are introduced and also, the well-applied methods in solving the subsequent problem in each step.

For the third step, crossing minimization, we introduced a new barycenter-based algorithm, we named it the efficient barycenter algorithm. A empirical study on the proposed efficient barycenter algorithm and the most recent state-of-the-art algorithm (global sifting) [BBBH11], besides some barycenter family algorithms, has indicated that the efficient barycenter algorithm produces better results and runs fast.

Finally, a new idea about the need of generating random hierarchical graphs is introduced proposing a mathematical form for the maximum number of edges in a hierarchical graph. Furthermore, a hierarchical graph (proper or generally non-proper) generator is considered controlling all the parameters of a hierarchical graph, like the number of layers, the number of vertices in each layer, the minimum number of outgoing edges of a vertex, the edge density and the ratio of the long edges in the graph.

Algorithm 3.13: Proper Hierarchical Graph Generator

Input : $\underline{k}$, $\bar{k}$, $\underline{n}$, $\bar{n}$, d^+, D.
Output: A *proper* hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k; \mathcal{E})$ with edge density D.

```
1  k = random generated number between k_low and k_high;
   // Creating the vertices in each layer
2  for i = 1 to k do
3      n_i = random generated number between n_low and n_high;
4      for j = 1 to n_i do
5          set the number of the j^th vertex in layer V_i be j;
   // Creating the edges
6  for i = 1 to k − 1 do
7      for j = 1 to n_i do
8          for s = 1 to d+ do
9              Choose a source vertex u ∈ V_i randomly;
10             Choose a target vertex v ∈ V_{i+1} randomly;
11             Insert an edge (u, v);
12             m = m + 1;
   // Checking isolated vertices in the last layer
13 for j = 1 to n_k do
14     Let v be the j^th vertex in layer V_k;
15     if d−(v) = 0 then
16         Choose a random source vertex u ∈ V_{k−1};
17         Insert an edge (u, v);
18         m = m + 1;
   // Inserting the rest edges according to graph density
19 Let the number of maximum edges m_max = Σ_{i=1}^{k−1} n_i × n_{i+1};
20 Let the number of the rest edges m_rest = (m_max × D) − m;
21 for i = 1 to m_rest do
22     Choose a random layer number s, 1 ≤ s ≤ k − 1;
23     Choose a random source vertex u ∈ V_s;
24     Choose a random target vertex v ∈ V_{s+1};
25     Insert an edge (u, v);
26     m = m + 1;
27 return (H);
```

Algorithm 3.14: General Hierarchical Graph Generator

Input : $\underline{k}$, $\bar{k}$, $\underline{n}$, $\bar{n}$, d^+, D, l^*.

Output: A not-necessarily proper hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k; \mathcal{E})$ with edge density D and long edge ratio l^*.

k = random generated number between $\underline{k}$ and $\bar{k}$;
`// Creating the vertices in each layer`
for $i = 1$ *to* k **do**
    n_i = random generated number between $\underline{n}$ and $\bar{n}$;
    for $j = 1$ *to* n_i **do** set the number of the j^{th} vertex in layer $\mathcal{V}_i$ be j;
`// Creating the edges`
for $i = 1$ *to* $k - 1$ **do**
    for $j = 1$ *to* n_i **do**
        for $p = 1$ *to* d^+ **do**
            Choose a random source vertex $u \in \mathcal{V}_i$;
            Choose a random layer number t, $i + 1 \leq t \leq k$;
            Choose a random target vertex $v \in \mathcal{V}_t$;
            Insert an edge (u, v);
            if $t = i + 1$ **then** $m_s = m_s + 1$; **else** $m_l = m_l + 1$;
`// Checking isolated vertices in the last layer`
for $j = 1$ *to* n_k **do**
    Let v be the j^{th} vertex in layer $\mathcal{V}_k$;
    if $d^-(v) = 0$ **then**
        Choose a random layer number s, $1 \leq t \leq k - 1$;
        Choose a random source vertex $u \in \mathcal{V}_s$;
        Insert an edge (u, v);
        if $t = i + 1$ **then** $m_s = m_s + 1$; **else** $m_l = m_l + 1$;
`// Inserting the long edges and the rest short edges`
Let the number of maximum edges $m_{max} = \frac{1}{2}\left(n^2 - \sum_{i=1}^{k} n_i^2\right)$;
Let the number of long edges $m_{l-rest} = (m_{max} \times l^*) - m_l$;
Let the number of the rest edges $m_{s-rest} = m_{max} - m_l^* - m_l - m_s$;
for $i = 1$ *to* m_{s-rest} **do**
    Choose a random layer s, $1 \leq s \leq k - 1$;
    Choose a random source vertex $u \in \mathcal{V}_s$;
    Choose a random target vertex $v \in \mathcal{V}_{s+1}$;
    Insert an edge (u, v); $m = m + 1$;
for $i = 1$ *to* m_{l-rest} **do**
    Choose a random layer number s, $1 \leq s \leq k - 2$;
    Choose a random source vertex $u \in \mathcal{V}_s$;
    Choose a random layer number t, $s + 2 \leq t \leq k$;
    Choose a random target vertex $v \in \mathcal{V}_t$;
    Insert an edge (u, v); $m = m + 1$;
$m = m_s + m_l$;
return *(H)*;

4

Dynamic Hierarchical Graph Drawing

Visualize this thing that you want, see it, feel it, believe in it. Make your mental blue print, and begin to build.[1]

In this chapter we study the problem of dynamic hierarchical graph drawing. We introduce a general framework for computing the difference between two drawings of a dynamic hierarchical graph. The proposed framework is generally enough to be applied to any hierarchical graph of any size and can also be extended to be applied to any graph type of any size. In the second part of this chapter, we discuss the problem of crossing minimization in the dynamic graph drawings of hierarchical graphs. This means, we want to select the drawing that minimizes the number of crossings and at the same time minimize the differences between that drawing and the original one.

4.1 Introduction

In many applications, graphs are *dynamic* since changing actions are executed on a graph in order to reflect the evolution of the system behaviour represented by that graph. Dynamic graph drawing scenarios are involving a repeated redrawing of the graph after some frequently occurring actions (changes) to the graph structure and/or some of its geometric characteristics. Some examples for the dynamic graph drawing scenarios are:

[1] Robert Collier.

- *Interactive user interfaces systems* [OW78, PST97, WB04] allow the user to manipulate the drawing of a graph and this necessarily needs an algorithm that should be capable of adapting to the dynamic changes. The system may allow the user to explicitly insert, delete, or modify vertices and/or edges as a part of a simulation, cluster and minimize some related vertices, changing some edge curve by modifying its source and/or target vertex, or simply modify the positions of some individual vertices as they like, such as vertex v should be on the right of vertex u.
- *Internet websites* on a server could be moved to another one according to some maintenance conditions [HA99a, HA99b]. Also, websites connectivity can change over time. Consequently this implies that the corresponding network representation for these websites changes.
- In *large graphs* drawings (such as social networks) [GW06, Sco00, MR10], it should be possible to scale (expand or collapse) parts of the drawing in order to focus on the connections and objects relations in this part. In case of expanding parts, those expanded parts are enlarged where other parts of the drawing will disappear behind the expanded parts.

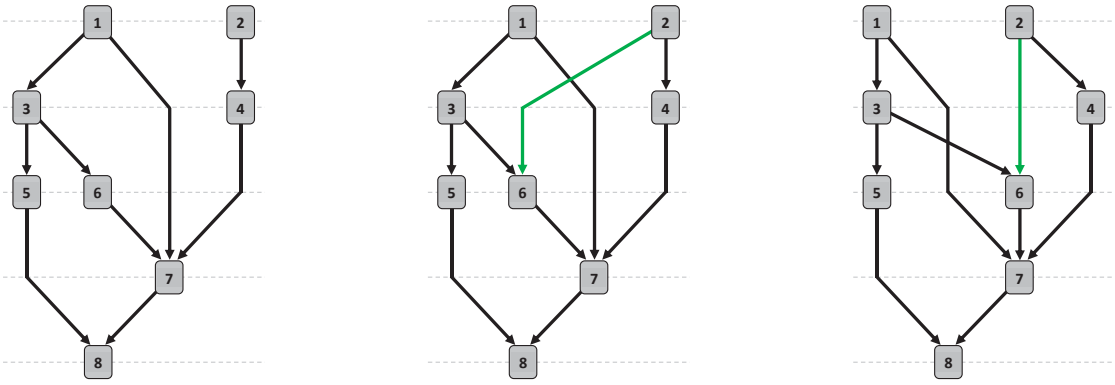
The simplest way to solve the above-mentioned dynamic environment situations is to consider the drawing of the graph after each change from scratch as an independent problem and then apply a static graph drawing algorithm to produce the new drawing after each action. Unfortunately, this approach has two main disadvantages: firstly, since the graph has been slightly modified, many existing computations of the old drawing will be recomputed again, which means that much time will be wasted in a redundant work. Secondly, and more important, since the user is actually familiar with the current drawing, he/she has to spend some effort and time to be re-familiarized with the new drawing after executing the change. The user here has really built a so-called user's "mental map" [ELMS91, MELS95], which should be preserved as much as possible. Consequently, this means in *dynamic graph drawing problems*, the new drawing should try to maintain the user's mental map besides usual static objectives (like minimize the drawing area) of the graph drawing. Sometimes, *dynamic stability* is another name of preserving the user's mental map. A number of authors addressed the dynamic graph drawing problem and devised algorithms that provide special treatment of preserving the mental map after a graph changes [Moe90, CDT+92, MELS95, PCJ96, Nor97, Pur97, BW97, LMR98, Pur98, BT00, Bra01, Pur02, EGK+03, EKLN04, GBPD04, PKL04, FT04, HEL05, KG06, GW06, SQ07, PB08, APP11b, DHW11, BVB+11, BM12].

This chapter is divided into 4 sections of which this section is the first. Section 4.2 contains two issues: the first one contains the basic concepts of the mental map and the different techniques for its preservation. The second introduces some mathematical formulations for the evaluation of a change on a graph drawing. Then, we go deeply into presenting the problem of dynamic drawing of hierarchical graphs in Section 4.3 by presenting a general framework of computing the change between two different drawings of two different hierarchical graphs, then applying the proposed framework to some existing metrics used with drawing types rather than hierarchical drawing, and introducing some examples of the proposed difference metrics. Section 4.4 contains some initial attempt for combining the two problems of dynamic drawing of hierarchical graphs and minimizing edge crossings of hierarchical graphs.

4.2 Preserving the Mental Map

The concept of mental map is intuitive. In a dynamic graph drawing environment, when a user looks at a drawing, he or she will learn about the drawing structure, how to navigate through the drawing and try to understand the relations between the drawing components. The effort to become familiar with a drawing has been termed "building mental map". Much work has been done in preserving the mental map [ELMS91,MELS95,PST97,DG02,GBPD04, PHG08, SP08, DLF+09]. As noted by Papakostas, Six, and Tollis in [PST97], "Obviously this is a waste of human resources to continually reanalyze the entire drawing and also of computational resources to recompute the entire drawing after each modification." Hence, the drawing must not only remain readable and be easy to be understand over time, but also the user's mental map must be preserved as much as possible.

The example shown in Figure 4.1 considers the previous note. Suppose the drawing in 4.1(a) is the current drawing, and the action is to insert a new edge connecting the two vertices 2 and 6. The drawing depicted in 4.1(b) is more similar to the original one in 4.1(a) since all the positions of the vertices are kept unchanged and all the edges keep their original routings. Where applying a graph drawing algorithm from scratch might produce the drawing presented in 4.1(c), which looks, at first impression, quite different from the original one in 4.1(a).



(a) (b) (c)

Figure 4.1: Current drawing (a) before executing any actions and two possible drawings (b) and (c) for inserting a new long edge (2,6). Drawing (b) is more similar to (a) than (c) since it keeps the positions of the vertices and the routings of the edges. So, drawing (b) preserves the mental map more than (c).

There are two suggested solutions of preserving the user's mental map problem:

1. Highlighting the changes by introducing the inserted or deleted vertices and edges in different colours (for a few seconds), vertices which are to be removed from the drawing to slowly fade out of view, or simply animating them. This could easily help the user to be aware of the changes and that the transitions between the two drawings are clearly recognized.

2. Minimize the changes in the new drawing such that the effort to user refamiliarity is minimized. For the example in Figure 4.1, drawing 4.1(b) keeps the position of all vertices of the original drawing 4.1(a), hence the graph could be immediately recognized as it is approximately the same graph as in 4.1(a). Actually, this objective contradicts with some traditional aesthetic criteria (such as minimization of edge crossings, or distributing the vertices uniformly etc.). Now, we have a trade-off between these two objectives and hence a compromise is required.

Recent computational advances have made the use of animation to show smooth changes between the current drawing and another much more widespread, see, for example, [TMB02, WB04, GMH+06, RFF+08, APP11a]. This is a significant advantage in allowing the user's knowledge of a previous state of a graph drawing to transfer to the new one, since the user can track the incremental movement of vertices and the routing of edges over time. Obviously, animation seems to be relatively straightforward and will not be discussed here in more detail. Instead, we focus in this chapter on the second issue, which is change minimization.

Sometimes, an animation of a dynamic graph is built "*off-line*", meaning that the evolution of the graph is already known in advance and the task is to stitch together snapshots of the network previously taken at various intervals. In this case, rather than taking each new drawing change in isolation, we can take a higher level approach and aim to minimize disruption to the mental map throughout the entire animation. Another scenario is the *on-line* dynamic graph drawing, in which the original graph $\mathcal{G}_0$ with drawing $\mathcal{D}_0$, a series of s actions $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_s$ produce a set on s graphs $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_s$, where the actions are not known in advance. The objective here is to produce s drawings $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_s$ such that $\mathcal{D}_i$ is the corresponding drawing of the graph $\mathcal{G}_i$, $1 \leq i \leq s$, such that the difference between each one of the new drawings and the original one $\mathcal{D}_0$ is minimized.

This requires to clarify the intuitive but rather fuzzy meaning of *minimizing changes to a drawing* in a way that the mental map is highly preserved, an effort that has also been termed *maintaining dynamic stability*. So far, numerous models have been suggested in the literature to capture the notion of the mental map. These models can be grouped into two basic categories: either the allowed changes are restricted to a subset of the vertices, or execute the action freely and then some mathematical difference metrics to measure the value of change are used. The second category allows to trade-off aesthetic objectives with change. These two general approaches will be treated in more detail in the following subsections.

Bridgeman and Tamassia [BT00] systematically examined and compare a set of mathematical difference metrics for orthogonal drawings, where most of them can be applied to other drawing styles. We have actually updated some of their mathematical models in computing the difference in hierarchical drawings. Also, Purchase [Pur02] introduced an attempt to create a quantifiable method for assessing the aesthetic quality of any graph drawing, where mathematical metric formulation for seven common aesthetic criteria have been defined. These metrics are useful both for formally analysing the aesthetic quality of graph drawings produced by different algorithms, and for measuring the status of intermediate drawings produced in iterative drawing methods like genetic algorithms. The metrics are continuous, and can, therefore, be used to investigate the extent to which a drawing needs to conform to an aesthetic, rather than always insisting on an extreme. Many user studies for examining relevant aesthetic criteria have been introduced [PCJ96, Pur97, Pur00, Pur00, PMCC01, BT01, PCA02, PHG08, Hua07, SP08, PS08, HvW09, DLF+09, FQ11, ZKS11, APP11b, PPP12].

4.2.1 Restricting Changes

An ideal concept of preserving the mental map is to not allow any changes to the current locations of the vertices and routes of the edges that are not directly affected by an action. Then, the drawing algorithm only computes the locations of new vertices and depicts the routing of the new edges, which is done in a way to minimize common traditional aesthetic criteria. Although this approach perfectly maintains the mental map, the resulting drawing may be quite bad according to other aesthetic criteria, because, e.g., many edge crossings or edge bends usually could not be avoided.

Restricting the changes just to a small set of vertices and edges may be weakened by allowing adjustment of vertices in the *vicinity* of an action. Böhringer and Paulisch [BP90] defined the vicinity of an action as all vertices directly affected by that action as well as vertices with a distance smaller than a certain edge length. This completely reflects the idea that *a user may tolerate changes in a small portion of the drawing around the area where the graph structure changed, but would prefer the remainder of the drawing to stay fixed.* We introduce a related definition of the vicinity of an action to be just the set of vertices and edges that will be inserted to or deleted from the current drawing. Afterwards, we build a general mathematical form of a difference metric based on this definition. The details of this assumption are introduced in Section 4.3.

Restricting the set of vertices that may be adjusted after a change is particularly useful for large graphs, since it also reduces the execution time of the considered algorithm and also for redrawing the new drawing.

4.2.2 Difference Metrics

Many authors suggested another approach to preserve the mental map instead of attempting fixing parts of the drawing. This approach based on defining some difference metrics, with mathematical formulations, to measure the similarity (or dissimilarity) between the current and the new drawings in order to evaluate the effort required to rebuild the mental map after executing an action. The objective here is to measure precisely the value of changes in a drawing when certain updating changes are executed. The role of an algorithm now is to produce a new drawing that has a good compromise between the considered traditional aesthetic criteria and the similarity of the new drawing to the current one. Note that computing the similarity between two drawings (or generally, two shapes) or drawing graphs symmetrically is a relative problem and has received a lot of attention [Man91, BT01, AHT02, CY02, BJ03, PKL04, HL06, BL10].

An important advantage of this approach is that it allows arbitrary changes to the current drawing in order to guarantee the rules of the drawing style. On the other hand, to find a good trade-off between traditional aesthetic criteria and preserving the mental map may be difficult. "Also, it does not yet seem to be clear how to actually measure similarity with respect to the mental map" [Bra01, page 223]. Also, combining traditional aesthetic criterion (like minimizing area or reduce crossings) in a dynamic drawing environment represents an interesting multi-objective optimization problem that can be solved using metaheuristic techniques like evolutionary algorithms or the ant colony technique [Deb01, DF07].

In the following subsections, a set of the suggested difference metrics are grouped according to the general concept they are based on.

4.2.2.1 Distance-Based Metrics

The *distance* metrics [BT00] reflect the simple intuition that drawings look very different if the vertices cannot be aligned very well. Since vertices alignment is based on distance minimization, distance metrics basically measure the quality (or inferior) of the vertices alignment. In order to make the value of the distance metrics comparable between pairs of vertices, they should be scaled by the drawing unit length $\mathcal{U}$. Some examples of distance metrics are *Hausdorff distance*, *Euclidean distance* and *relative distance* metrics. Euclidean distance and relative distance metrics are introduced in Sections 4.3.4 and 4.3.5 with an extension to be applied to hierarchical drawings.

4.2.2.2 Proximity Metrics

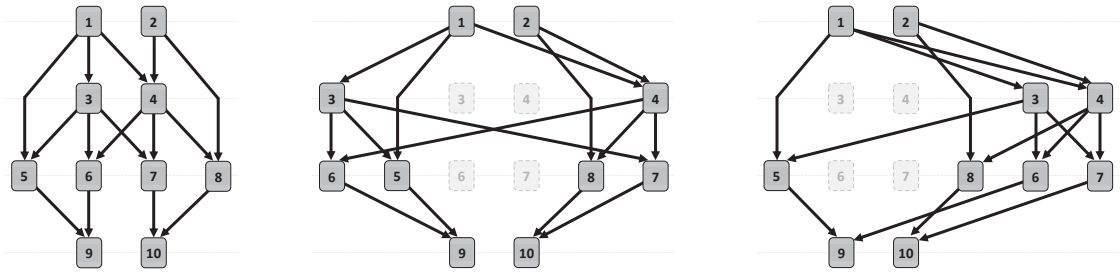
The *proximity* (or *clustering*) metrics reflect the idea that vertices near to each other in the first drawing should remain near to each other in the second drawing. This is stronger than the distance metrics since it captures the idea that if an entire subgraph moves (such that there are no changes within the vertices of that subgraph) relative to another subgraphs, the distance should be less than if each vertex in one of the subgraphs moves in a different direction. See Figure 4.2 as an example.

Basically any of the methods that are used to find the clustering of a graph may be considered as proximity metric. The general idea here is to use some proximity relation of the clusters and compare the relationship between the set of vertices of the current graph before and after the layout change. There are different proximity metrics are considered such as: *sphere of influence graph* [ELMS91], *Delaunay triangulation* [Lyo92, MELS95, LMR98], *ϵ-clustering* [ELMS91, BT00], *nearest neighbour within* [LMR98, BT00], *nearest neighbour between* [LMR98, BT00].

4.2.2.3 Partitioning Metrics

Instead of capturing the vertices individually as in distance and proximity metrics, *partitioning* metrics are introduced [BT00]. The partitioning metrics are based on dividing the vertices into smaller parts according to some criteria, and then measuring qualities of these parts. The intuition for the partitioning metric is to monitor "visual units" that the user may use for capturing the new drawing.

The partitioning method and metrics can be computed quite simply. Firstly, the vertices set can be divided so that the vertices in each part have the same relative position in both drawings. This identifies blocks of the drawing that are the same in both drawings. Note that the larger partitions size, the more unchanged parts and the more similar the drawings. Then, the partition metric can be computed by considering the average partition size or the number of partitions.



(a) (b) (c)

Figure 4.2: Proximity example: drawing (c) is similar to drawing (a) more than drawing (b) because the relative distance between the four vertices 3, 4, 6 and 7 is smaller. Original alignment of the four vertices in drawing (a) is shown in light grey colour in the two drawings (b) and (c).

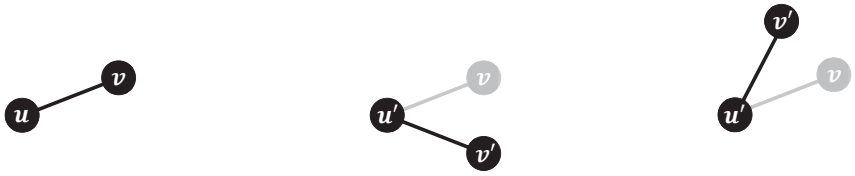
4.2.2.4 Orthogonal Ordering Metrics

The *orthogonal ordering* metric reflects the desire to preserve the relative ordering of every pair of vertices. If v is north-east of u in drawing $\mathcal{D}$, v' should remain to the north-east of u' in the new drawing $\mathcal{D}'$ [ELMS91, MELS95]. The simplest measurement of difference in the orthogonal ordering is to take the angle between the vectors $v - u$ and $v' - u'$ (constant-weighted orthogonal ordering). This has the desirable feature that if v is far from u, $d(v, u)$ must be larger in order to result in the same angular move, which reflects the intuition that the relative position of points near each other is more important than the relative position of points that are far apart. This problem can be addressed by introducing a weight that depends on the particular angles involved in the move in addition to the size of the move (*linear-weighted orthogonal ordering*). This is because using the angular change fails to take into account situations like that in Figure 4.3.

Lyons, Meijer, and Rappaport [LMR98] used the λ-matrix model for measuring the difference of two point sets. The λ-matrix model is based on the concept of order type of a point set introduced by Goodman and Pollack [GP83]. This model tries to capture the notion of the relative position of vertices in a straight-line drawing. For a graph with n vertices, a $n \times n$ matrix M is computed. In this matrix M, each entry (i, j) contains the number of points that lie on the left of the directed line from vertex i to vertex j. Then, the difference metric is computed as the sum of the differences in all the entries of M before and after a change.

4.2.2.5 Shape Metrics

The *shape* (or *edge routing*) metric is motivated by the reasoning that edge routing may have an effect on the overall look of the graph drawing in case of the positions of the vertices are the same in both drawings. This metric is based on the intuition that the position of vertices is more important than the routing of the edges because vertices are remembered as locations, while edges are traced "on the fly" to discover connections between vertices [Nor97].



(a) (b) (c)

Figure 4.3: Orthogonal ordering example: although the angle the vertex v moves relative to vertex u is the same from (a) to (b) and to (c), the perceptual difference between (a) and (c) is much greater. Note that u' and v' are the representations of the two vertices u and v in drawing $\mathcal{D}'$. Also, the original location of vertex v in drawing $\mathcal{D}$ is shown in grey colour in the two drawings (b) and (c).

In the case of orthogonal drawing, Bridgeman and Tamassia [BT00] assumed the shape of an edge as the sequence of directions (north, south, east, and west) travelled when traversing the edge. The edge routing is used as distance metric and computed by comparing the sequence of directions for each edge and then measuring the number of edit operations (insert, delete or replace) to transform the old sequence of directions into the new one.

For non-orthogonal edges the direction is taken to be the most prominent direction. For example, if the edge goes from (1,1) to (4,2), then the most prominent direction is east. For each pair of edges (e_i, e_i'), the edit distance between the corresponding shape strings is computed. To determine the edit distance, two algorithms are used. The first one uses *dynamic programming* to compute the minimum number of insertions, deletions, or replacements of characters required to transform one string into the other. The second is similar, but normalizes the measure according to the length of the strings using a techniques introduced by Marzal and Vidal [MV93]. After that, the value of the shape metric is computed as the average edit distance over the edges of the graph.

Further suggestions to capture the notion of the mental map include to maintain congruence [ELMS91] which only allows the operations reflection, rotation, translation and scaling, or to maintain topology, i.e. the dual graph of a layout [MELS95]. Furthermore, a great deal of work has been done on point set matching. For several methods of obtaining both optimal and approximate matching, see, for example, [ISI89, CGH+97, GMO99].

4.2.2.6 Identical Drawing Pleasing Metrics

The measurement of aesthetic criteria within a drawing of a graph is done informally, and normally differs according to the used algorithm. Also, there is no standard, objective way for analysing a drawing with respect to the sense of different geometric aesthetics. Furthermore, continuous measurements of metrics are necessary, so that analysing a drawing with respect to an aesthetic is not merely a binary decision, for example, considering a drawing to be "*planar*" or "*not planar*", but is rather an indication of the extent to which the drawing conforms to the aesthetic (for example, a drawing may be considered to have 25% presence of crossing, i.e. 90% planar).

An important advantage of providing a formal method for analysing the aesthetic quality of drawings, is that these computational aesthetic metrics can be used for the definition of cost functions for some metaheuristic techniques. So, an ideal aesthetic quality of a drawing can be defined in advance (for example, the drawing should have at least 80% symmetry, at most 10% "crossings", and 15% "edge orthogonality"). Afterwards, some evaluation function, which determines whether these criteria are satisfied or not, may then be implemented using computational metrics for each of these aesthetics, and can also be used to indicate whether more iterations are needed or not.

Many algorithms try to get the extreme of the aesthetics, for example, removing all bends, or ensuring that all vertices are placed on an invisible grid. Although there is no doubt that these aesthetics criteria improve the readability of graph drawings, the usefulness of an aesthetic is related to a critical mass rather than an extreme: perhaps drawings with at most 10% "crossings" are as useful as those with 0% "crossings" or perhaps a drawing needs to be at least 90% orthogonal before the usability effects of the orthogonality aesthetic are evident. Without continuous measures, this notion of critical mass, and the computational implication of relaxing the requirement that the aesthetic be satisfied at the extreme cannot be investigated.

Purchaise [Pur02] presented some formal metrics for measuring the aesthetic presence in an identical drawing of a graph for seven common aesthetic criteria. These seven aesthetic criteria are: minimizing edge crossings, minimizing edge bends, maximizing drawing symmetry, maximizing the minimum angle between the adjacency edges of a vertex, maximizing edge orthogonality, maximizing vertex orthogonality and maximizing consistent flow direction (only for directed graphs). These metrics are applicable to any graph drawing of any size. The metrics are useful for determining the aesthetic quality of a given graph drawing. Most of these metrics are presented in the next section and are also extended to be applied to hierarchical drawing.

4.3 Difference Metrics for Hierarchical Graphs

Bridgeman and Tamassia [BT00] defined some metrics associated with graph drawings. Their concern was measuring the differences between two drawings of the same graph in a dynamic environment. These metrics are applied to orthogonal drawings of graphs. Purchase [Pur02] introduced formal metrics for measuring the aesthetic presence of a drawing of a graph for seven common aesthetic criteria, applicable to any graph whatever the graph size. These metrics are intended to be applicable in the analysis of drawings of any hierarchical graph of any structure or size, and enabling quantitative comparisons between drawings of different hierarchical graphs.

Significant progress has been made in drawing dynamic clustered graphs [FT04], drawing dynamic trees [Moe90], dynamic planar graphs [DT89] and dynamic series-parallel graphs [CDTT95]. Although these are useful techniques for these specific graph types, they could not be applied to general graph drawing. Hornick, Miriyala and Tamassia [MHT93] described a practical incremental edge router for orthogonal drawings, such as entity-relation diagrams.

The Sugiyama heuristic [STT81] and some variants of it are quite popular for drawing hierarchical graphs. Böhringer and Paulisch [BP90] demonstrated how Sugiyama heuristic could be modelled in terms of constraints, and then transformed to preserve dynamic stability. In the layering step, for each edge (u, v) a constraint is introduced assuming that vertex u should be placed above vertex v, and the layering is decided. Then, the barycenter ordering is used to derive constraints to determine the horizontal ordering of the vertices in each layer.

The *DynaDAG* system [Nor97] is an adaptation of the Sugiyama heuristic that allows interactive changes to the structure of a hierarchical graph. *DynaDAG* allows to insert, modify or delete a single vertex or edge. Vertices are originally placed on the highest possible layer and may be moved down when this becomes necessary by an insertion of another vertex or edge. Vertices are moved down layer by layer, shifted in each layer to their median position with respect to their adjacent vertices. When the vertices are in their final position, the adjacent edges are adjusted. New inserted edges are routed heuristically. The final vertex coordinates are calculated by a linear program. The placement of the user is taken into account when inserting a new vertex. A final point about *DynaDAG* system is that only distance metrics are used to preserve the mental map. So, for hierarchical graphs, it is motivated to apply other geometric metrics (like edge orthogonality, vertex minimum angle, ...) that actually applied in preserving the mental map of other graph types. North and Woodhull [NW02] propose a heuristic for dynamic hierarchical graph drawing using the *DynaDAG* system.

Branke [Bra01] suggested two constraints to preserve the user's mental map when a hierarchical graph is modified. These two constraints are derived from the old drawing determining:

1. the ordering of the vertices in each layer, and
2. the set of vertices belonging to the same layer in the old drawing should also belong to the same layer in the new drawing.

Then, only vertices close to the change in the graph (i.e. vertices in the vicinity of the change) are exempt from these two constraints and could move freely. By setting the size of the vicinity, the user mental map could be highly preserved. Given the total set of constraints (Sugiyama plus mental map preserving constraints), constraint propagation is used to find a feasible drawing. By assigning priorities to the constraints, inconsistencies are resolved by neglecting some of them.

In the next subsections, we introduce first our proposed general framework for computing the value of change between two drawing of two different hierarchical graphs. Afterwards, we apply our general framework to a set of difference metrics used in the literature with different graph types rather than hierarchical graphs.

In the following, we suppose that the considered hierarchical graphs are proper, the layers are drawn and numbered from top to bottom, and the long edges have been actually broken into chains of dummy vertices over the intermediate layers.

4.3.1 Dynamic Hierarchical Graph Actions

After executing an action $\mathcal{A}$ on a hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$, another hierarchical graph $\mathcal{H}' = (\mathcal{V}', \mathcal{E}', \phi')$ will be produced, i.e.:

$$\begin{aligned} \mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi) &\xrightarrow{\mathcal{A}} \mathcal{H}' = (\mathcal{V}', \mathcal{E}', \phi'), \text{or} \\ \mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E}) &\xrightarrow{\mathcal{A}} \mathcal{H}' = (\mathcal{V}'_1, \mathcal{V}'_2, \cdots, \mathcal{V}'_k; \mathcal{E}') \end{aligned}$$

Here, as in some of the literature (as an example [Nor97]), we assume that the executed action $\mathcal{A}$ belongs to the product:

$$\{ \textit{insert} , \textit{delete} , \textit{modify} \} \times \{ \textit{vertex} , \textit{edge} \} \tag{4.1}$$

More complex actions must be decomposed of basic primitives. This because applications often require performing a number of actions (updates) at once, since from the user point of view, it would make sense to collect updates and execute them together at once. The *modify vertex* action means moving a vertex vertically to a different layer (if it possible) or moving a vertex horizontally by changing its position in its layer. The *modify edge* action means changing the edge source or target vertex.

Depending on the topological structure of hierarchical graphs, the following 13 actions represent the different cases of an action that could be executed on a hierarchical graph:

1. Insert a new *short edge* (u, v) where $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_{i+1}$. An example of this action is introduced in Figure 4.10.
2. Insert a new *long edge* (u, v) where $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ such that $j > i + 1$. This implies introducing a set of $j - i - 1$ dummy vertices over the $j - i - 1$ intermediate layers $\mathcal{V}_{i+1}, \mathcal{V}_{i+2}, \cdots, \mathcal{V}_{j-1}$. An example of this action is introduced in Figure 4.11.
3. Delete an existing *short edge* (u, v) where $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_{i+1}$. An example of this action is introduced in Figure 4.12.
4. Delete an existing *long edge* (u, v) where $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ such that $j > i + 1$. This implies deleting the set of $j-i-1$ dummy vertices over the $j-i-1$ intermediate layers $\mathcal{V}_{i+1}, \mathcal{V}_{i+2}, \cdots, \mathcal{V}_{j-1}$. An example of this action is introduced in Figure 4.13.
5. Modify an existing *short/long edge* (u, v) where $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ such that $j \geq i + 1$. This means changing the source vertex u or the target vertex v. An example of this action is introduced in Figure 4.14.
6. Insert a new *vertex* v into layer $\mathcal{V}_i$ connecting two vertices $u \in \mathcal{V}_{i-1}$ and vertex $w \in \mathcal{V}_j$, $j \geq i + 1$. This consequently implies inserting two new edges: a short edge (u, v) and a short/long edge (v, w). An example of this action is introduced in Figure 4.15.
7. Insert a new *vertex* v into a new *first* layer $\mathcal{V}_0$ connecting vertex $w \in \mathcal{V}_1$. This consequently implies inserting a new short edge (v, w). An example of this action is introduced in Figure 4.16.
8. Insert a new *vertex* v into a new *last* layer $\mathcal{V}_{k+1}$ connecting vertex $u \in \mathcal{V}_k$. This consequently implies inserting a new short edge (u, v). An example of this action is introduced in Figure 4.17.
9. Insert a new *vertex* v into a new *intermediate* layer $\mathcal{V}_i$ connecting two vertices $u \in \mathcal{V}_{i-1}$ and vertex $w \in \mathcal{V}_{i+1}$. This consequently implies inserting two new short edges (u, v) and (v, w). An example of this action is introduced in Figure 4.18.

10. Delete an existing *vertex* v from layer $\mathcal{V}_i$. This consequently implies deleting the adjacency edges of v. This means that layer $\mathcal{V}_i$ has at least one non-dummy vertex rather than v. An example of this action is introduced in Figure 4.19.

11. Delete an existing *vertex* v and its layer $\mathcal{V}_i$. This means that all the vertices (if exist) in layer $\mathcal{V}_i$, rather that v, are dummy vertices. This consequently implies deleting the adjacency edges of v and any the long edges (u, w) such that $u \in \mathcal{V}_{i-1}$ and $w \in \mathcal{V}_{i+1}$ will be modified to a short one. An example of this action is introduced in Figure 4.20.

12. Modify the *horizontal position* of an existing *vertex* $v \in \mathcal{V}_i$ by changing its order (or its x-coordinate) in its layer. An example of this action is introduced in Figure 4.21.

13. Modify the *vertical position* of an existing *vertex* $v \in \mathcal{V}_i$ by moving it to an upward to some layer $\mathcal{V}_j$, $1 \leq j < i$ or down to layer $\mathcal{V}_q$, $i < q \leq k$. This action means that all edges (if exist) incident on or go out from v are long edges. This means modifying the adjacency edges of v. An example of this action is introduced in Figure 4.22.

The geometric difference between two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ of two different graphs $\mathcal{H}$ and $\mathcal{H}'$, is not completely reflected by the arithmetic difference ($|\mu(\mathcal{D}) - \mu(\mathcal{D}')|$) between the two metric values of the two identical drawings $\mu(\mathcal{D})$ and $\mu(\mathcal{D}')$, as presented in [Pur02]. Consider the following example. The *edge orthogonal angle* represents how far away the edge segment deviates from an orthogonal angle and the *edge orthogonality metric* is computed as the average of the orthogonal deviations of the set of edges in the graph, i.e. the sum of the edges orthogonal deviation of the edges over the number of edges. Suppose the action to be executed is to insert a new short edge. Hence, the number of edges is increased (by 1) and also the sum of the orthogonal deviations of the edges is increased (by the orthogonal deviation value of the inserted edge). Consequently, the difference between the two edge orthogonality metrics is not precisely reflected the geometric difference between the two identical drawings.

So that, we introduce a new general framework of computing the actual geometric change between the two drawings. To do this, it is better to focus on and compute the change in each graph component (vertex/edge) in both drawings simultaneously, instead of just comparing the two metric values of the two identical drawings.

4.3.2 Shared and Action Components

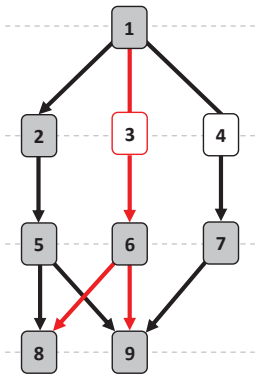
After executing an action, according to the product in Equation 4.1, on a hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$, a new hierarchical graph $\mathcal{H}' = (\mathcal{V}', \mathcal{E}', \phi')$. We define the *action components* as the components (vertices/edges) that will be inserted into or deleted from the hierarchical graph $\mathcal{H}$, where the remaining components (vertices/edges) will be considered as the *shared components* since they will be found in both graphs $\mathcal{H}$ and $\mathcal{H}'$. The value of any geometric difference metric will be computed as the combination of the value for the shared components and the action components.

Suppose $\mathcal{V}_a$ and $\mathcal{E}_a$ are the two sets of action vertices and action edges in graph $\mathcal{H}$ respectively, $\mathcal{V}'_a$ and $\mathcal{E}'_a$ are the two sets of action vertices and action edges in graph $\mathcal{H}'$ respectively, and $\mathcal{V}_s$ and $\mathcal{E}_s$ are the two sets of shared vertices and shared edges in the two graphs $\mathcal{H}$ and $\mathcal{H}'$ respectively. The relations between the shared and action components could be induced from the following formulas:

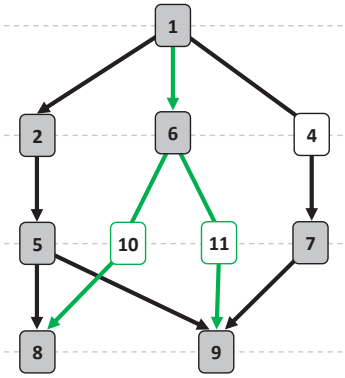
$$\left.\begin{array}{rcl} & \mathcal{V}_a & \subseteq \mathcal{V}, \\ & \mathcal{V}'_a & \subseteq \mathcal{V}', \\ & \mathcal{E}_a & \subseteq \mathcal{E}, \\ & \mathcal{E}'_a & \subseteq \mathcal{E}', \\ & \mathcal{V}_s & = \mathcal{V} \cap \mathcal{V}' = \mathcal{V} \setminus \mathcal{V}_a = \mathcal{V}' \setminus \mathcal{V}'_a, \\ & \mathcal{E}_s & = \mathcal{E} \cap \mathcal{E}' = \mathcal{E} \setminus \mathcal{E}_a = \mathcal{E}' \setminus \mathcal{E}'_a, \\ \Longrightarrow & \mathcal{H} & = (\mathcal{V}_a \cup \mathcal{V}_s, \mathcal{E}_a \cup \mathcal{E}_s, \phi), \text{ and} \\ \Longrightarrow & \mathcal{H}' & = (\mathcal{V}'_a \cup \mathcal{V}_s, \mathcal{E}'_a \cup \mathcal{E}_s, \phi') \end{array}\right\} \tag{4.2}$$

Figure 4.4 presents an example of the relation between the shared and action components. In this example, a hierarchical graph $\mathcal{H}$ drawn in Figure 4.4(a) 9 vertices in 4 layers and 9 edges where two edges are long and 7 are short. The action here is to move vertex number 6 one layer up. After moving vertex 6 one layer up, the two short edges (6,8) and (6,9) will be modified in the new drawing $\mathcal{D}'$ to long edges by inserting the two new dummy vertices 10 and 11. Furthermore, the long edge (1,6) will be modified to a short one in the new drawing $\mathcal{D}'$ and consequently the dummy vertex 3 will be deleted from the current drawing $\mathcal{D}$. So, for the two graphs $\mathcal{H} = (\mathcal{V}_a \cup \mathcal{V}_s, \mathcal{E}_a \cup \mathcal{E}_s, \phi)$ and $\mathcal{H}' = (\mathcal{V}'_a \cup \mathcal{V}_s, \mathcal{E}'_a \cup \mathcal{E}_s, \phi')$, where the shared and action components sets are:

$$\begin{array}{rcl} \mathcal{V}_a & = & \{3\}, \\ \mathcal{V}'_a & = & \{10, 11\}, \\ \mathcal{V}_s & = & \{1, 2, 4, 5, 6, 7, 8, 9\}, \\ \mathcal{E}_a & = & \{(1,3), (3,6), (6,8), (6,9)\}, \\ \mathcal{E}'_a & = & \{(1,6), (6,10), (6,11), (10,8), (11,9)\}, \text{ and} \\ \mathcal{E}_s & = & \{(1,2), (1,4), (2,5), (4,7), (5,8), (5,9), (7,9)\} \end{array}$$



(a) Current drawing $\mathcal{D}(\mathcal{H})$ before moving (modifying) vertex 6 one layer up, action components are in red colour



(b) New drawing $\mathcal{D}'(\mathcal{H}')$ after moving (modifying) vertex 6 one layer up, action components are in green colour

Figure 4.4: Example of action and shared components of hierarchical graphs.

4.3.3 General Difference Metric

The geometric aesthetic metrics measuring the similarity or dissimilarity between two different drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ of the two different graphs $\mathcal{H}$ and $\mathcal{H}'$ could be divided into two types: *vertex metrics* and *edge metrics*. Vertex metrics depend on the values of some geometric property of the vertices, while edge metrics depend on some geometric property of the edges. Examples of vertex metrics are Euclidean distance, relative distance, vertex minimum angle, and vertex size metrics. Examples of edge metrics are edge length, edge orthogonality and edge crossing metrics. These difference metrics will be presented in the next subsections.

Now, each graph consists of shared components and action components and the value of some metric $\mu(\mathcal{D}, \mathcal{D}')$ of the difference between the two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ could be divided into two parts. The first part μ_s is for the shared components and the second μ_a is for the action components, i.e:

$$\mu(\mathcal{D}, \mathcal{D}') = \mu_s + \mu_a \tag{4.3}$$

Suppose μ_V is a vertex metric, then the shared components will be the shared vertices, and then $\mu_s = \mu(\mathcal{V}_s)$ and $\mu_a = \mu(\mathcal{V}_a, \mathcal{V}'_a)$. Hence, the value of a vertex metric $\mu_\mathcal{V}$ is:

$$\mu_V(\mathcal{D}, \mathcal{D}') = \mu(\mathcal{V}_s) + \mu(\mathcal{V}_a, \mathcal{V}'_a)$$

Also, suppose $\mu_\mathcal{E}$ is an edge metric, then the shared components will be the shared edges, and then $\mu_s = \mu(\mathcal{E}_s)$ and $\mu_a = \mu(\mathcal{E}_a, \mathcal{E}'_a)$. Hence, the value of a vertex metric μ_E is:

$$\mu_E(\mathcal{D}, \mathcal{D}') = \mu(\mathcal{E}_s) + \mu(\mathcal{E}_a, \mathcal{E}'_a)$$

For the shared components, there are two representations for each shared component, one in each drawing of the two drawings $\mathcal{D}$ and $\mathcal{D}'$. So, the metric value of any shared component could be considered as the difference between the two metric values of that shared component in each drawing of $\mathcal{D}$ and $\mathcal{D}'$. Hence, the metric value μ_s for each shared component will be the summation of that metric value for all shared components. For the set of shared vertices $\mathcal{V}_s$, the change in a metric $\mu(\mathcal{V}_s)$ is:

$$\mu(\mathcal{V}_s) = \sum_{v \in \mathcal{V}_s} \left|\mu(r(v)) - \mu(r'(v))\right| \tag{4.4}$$

where $r(v)$ and $r'(v)$ are the geometric representations of the shared vertex $v \in \mathcal{V}_s$ in both drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively. Also, the change in a metric $\mu(\mathcal{E}_s)$ for the shared edges will be:

$$\mu(\mathcal{E}_s) = \sum_{e \in \mathcal{E}_s} \left|\mu(r(e)) - \mu(r'(e))\right| \tag{4.5}$$

where $r(e)$ and $r'(e)$ are the representations of the shared edge $e \in \mathcal{E}_s$ in both drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively.

For the action components, there is only one representation for each action component in just one drawing of the two drawings $\mathcal{D}$ and $\mathcal{D}'$. Consequently, the metric value μ_a of the action components could be considered as the difference between the two sums of the action components values in drawing $\mathcal{D}$ and that of the action components values in drawing $\mathcal{D}'$.

So, for the action vertices, the metric value $\mu(\mathcal{V}_a)$ for the action vertices could be considered as:

$$\mu(\mathcal{V}_a, \mathcal{V}'_a) = \left| \sum_{v \in \mathcal{V}_a} \mu(r(v)) - \sum_{v \in \mathcal{V}'_a} \mu(r(v)) \right| \tag{4.6}$$

where $r(v)$ is the representation of the action vertex $v \in \mathcal{V}_a$ or $v \in \mathcal{V}'_a$ in both drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively. Also, the change in a metric $\mu(\mathcal{E}_a)$ for the action edges will be:

$$\mu(\mathcal{E}_a, \mathcal{E}'_a) = \left| \sum_{e \in \mathcal{E}_a} \mu(r(e)) - \sum_{e \in \mathcal{E}'_a} \mu(r(e)) \right| \tag{4.7}$$

where $r(e)$ is the representation of the action edge $e \in \mathcal{E}_a$ or $e \in \mathcal{E}'_a$ in both drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively. Hence, by combining Equation 4.4 with Equation 4.6, the total change in a difference metric of the vertices μ_V will be:

$$\mu_V(\mathcal{D}, \mathcal{D}') = \sum_{v \in \mathcal{V}_s} |\mu(r(v)) - \mu(r'(v))| + \left| \sum_{v \in \mathcal{V}_a} \mu(r(v)) - \sum_{v \in \mathcal{V}'_a} \mu(r(v)) \right| \tag{4.8}$$

And, by combining Equation 4.5 with Equation 4.7, the total change in a difference metric of the vertices μ_E will be:

$$\mu_E(\mathcal{D}, \mathcal{D}') = \sum_{e \in \mathcal{E}_s} |\mu(r(e)) - \mu(r'(e))| + \left| \sum_{e \in \mathcal{E}_a} \mu(r(e)) - \sum_{e \in \mathcal{E}'_a} \mu(r(e)) \right| \tag{4.9}$$

All the difference metrics considered here are defined such that the measurement is a real number in [0,1], where 0 indicates that the two drawings are completely similar (this means that it is very easy to be refamiliarized with the new drawing and the mental map is highly preserved) where 1 assumes that the two drawing are completely dissimilar. Scaling the metrics in this way ensures that the metric value does not depend on the nature of the underlying the drawing style of the graph.

In order to get a general normalization for the computed change value between 0 and 1, we compare the value of change against some maximum value. By the definition of the user's mental map, after executing a change on the current drawing $\mathcal{D}$, which is the drawing that the user is actually familiar with, a new drawing is produced. So, it is reasonable to compare the value of difference between the two drawing $\mathcal{D}$ and $\mathcal{D}'$, according to some metric, against a maximum value $\mu_{max}(\mathcal{D})$ of the considered metric in the old drawing $\mathcal{D}$. Hence, after executing a change on a drawing $\mathcal{D}$ and producing a new one $\mathcal{D}'$, the final relative value of change with respect to some metric μ is:

$$\mu(\mathcal{D}, \mathcal{D}') = \frac{\mu_s + \mu_a}{\mu_{max}(\mathcal{D})} \tag{4.10}$$

Hence, the final normalized value of a vertex metric μ_V and edge metric μ_E are represented in Equation 4.11 and Equation 4.12 respectively.

$$\mu_V(\mathcal{D}, \mathcal{D}') = \frac{\sum_{v \in \mathcal{V}_s} |\mu(r(v)) - \mu(r'(v))| + \left| \sum_{v \in \mathcal{V}_a} \mu(r(v)) - \sum_{v \in \mathcal{V}'_a} \mu(r(v)) \right|}{\mu_{V max}(\mathcal{D})} \tag{4.11}$$

$$\mu_E(\mathcal{D},\mathcal{D}') = \frac{\sum_{e\in\mathcal{E}_s} |\mu(r(e)) - \mu(r'(e))| + \left|\sum_{e\in\mathcal{E}_a} \mu(r(e)) - \sum_{e\in\mathcal{E}'_a} \mu(r(e))\right|}{\mu_{Emax}(\mathcal{D})} \tag{4.12}$$

Note that, the general difference metrics produced in Equation 4.10 could also be used in computing the similarity between two drawings regardless of preserving the mental map, i.e. when the user is not familiar with any drawings, but he/she just watches both drawing at the same time and does not care about the change action. The only difference will be that the denominator will not be considered as the maximum value of the first drawing D, it will be considered as the maximum values of both maximums in both drawings. This is because it is not guaranteed that the maximum metric value of the first drawing D will keep a normalization between 0 and 1 for the final difference metric value, and in order to control this gap, the maximum of the two maximums is used. Hence, the final normalized difference metric $\mu_S(\mathcal{D},\mathcal{D}')$ that compute a degree of similarity or dissimilarity between the two drawings $\mathcal{D}$ and $\mathcal{D}'$ of the two hierarchical graphs $\mathcal{H}$ and $\mathcal{H}'$ is:

$$\mu_S(\mathcal{D},\mathcal{D}') = \frac{\mu_s + \mu_a}{\max\{\mu_{max}(\mathcal{D}), \mu_{max}(\mathcal{D}')\}} \tag{4.13}$$

In the next subsections, we apply the two equations 4.11 and 4.12 to update the existing metrics applied to different graph types in order to compute the difference between two drawings of two different hierarchical graphs. Also, we normalize the computed metrics between 0 and 1 in order to monitor and scale the behaviour of the computed metrics.

4.3.4 Euclidean Distance Metric

A straightforward method for measuring similarity (or dissimilarity) between two drawings $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}'(\mathcal{G})$ of the same graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is to use the *Euclidean distance* metric. The *Euclidean distance* metric computes the average of the Euclidean distance each vertex moves from the first drawing to the second [BT00]. The idea is motivated by the notion that if shared vertices move a long way from their original positions in the first drawing $\mathcal{D}$, the second drawing $\mathcal{D}'$ will look very different. The *Euclidean distance* metric was originally introduced by Lyons, Meijer, and Rappaport [LMR98].

For two drawings $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}'(\mathcal{G})$ of the same graph $\mathcal{G}$, a matched point set $\mathcal{P}$ is the set of pairs (p_i, p'_i) where p_i is the location of the point number i in drawing $\mathcal{D}(\mathcal{G})$ and p'_i is the location of the point number i in drawing $\mathcal{D}'(\mathcal{G})$. Let the Euclidean distance $d(p_i,p_j)$ between two points p_i and p_j in the plane is $d(p_i,p_j) = \sqrt{|x(p_i) - x(p_j)|^2 + |y(p_i) - y(p_j)|^2}$. The *Euclidean distance* metric is represented as:

$$\frac{1}{\mathcal{U} \times |\mathcal{P}|} \sum_{1\le i\le|\mathcal{P}|} d(p_i, p'_i) \tag{4.14}$$

where $\mathcal{U}$ represents the drawing unit.

For the two hierarchical graphs $\mathcal{H} = (\mathcal{V}_s \cup \mathcal{V}_a, \mathcal{E}_s \cup \mathcal{E}_a, \phi)$ and $\mathcal{H}' = (\mathcal{V}_s \cup \mathcal{V}'_a, \mathcal{E}_s \cup \mathcal{E}'_a, \phi')$, we can apply the Euclidean distance metric to compute the average Euclidean distance *only* for the set of shared vertices $\mathcal{V}_s$. This is because each shared vertex has two geometric representations, one in each drawing of $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$. By applying Equation 4.14 to the set of shared vertices we get the following:

$$\frac{1}{\mathcal{U} \times |\mathcal{V}_s|} \sum_{v \in \mathcal{V}_s} d(l(v), l'(v))$$

where $l(v)$ and $l'(v)$ are the locations of any shared vertex $v \in \mathcal{V}_s$ in both drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ respectively.

In order to normalize the Euclidean distance difference metric between 0 and 1, we could not just consider the previous form of the average of the Euclidean distance each shared vertex moves, since it is not guaranteed for this average value to be between 0 and 1. However, we should compare the summation of the Euclidean distance each shared vertex v moves against some maximum possible distance. In hierarchical graphs, any shared vertex $v \in \mathcal{V}_s$ could move horizontally through the horizontal line presenting the layer that contains v and vertically through consecutive layers. Suppose x_{min} and x_{max} are the minimum and maximum x-coordinate in the drawing $\mathcal{D}(\mathcal{H})$ respectively, the maximum horizontal distance Δx_{max} could vertex v move is:

$$\Delta x_{max}(v) = max\{x(l(v)) - x_{min}\ ,\ x_{max} - x(l(v))\}$$

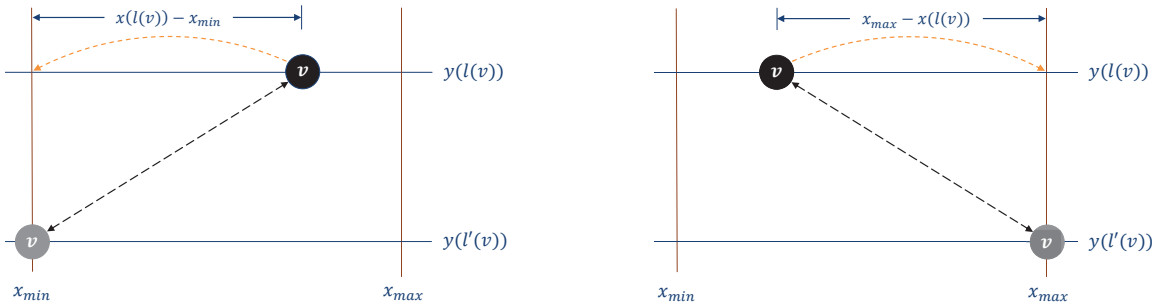
where the maximum vertical distance between the two locations $l(v)$ and $l'(v)$ of a shared vertex v in the two drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively is $|y(l(v)) - y(l'(v))|$. Figure 4.5 shows the maximum distance a shared vertex $v \in V_s$ can move. So, the Euclidean distance metric $\mu_{\mathcal{ED}}$ of the set of shared vertices in two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ of the two different hierarchical graphs $\mathcal{H}$ and $\mathcal{H}'$ respectively is:

$$\mu_{\mathcal{ED}}(\mathcal{D}, \mathcal{D}') = \frac{\sum_{v \in \mathcal{V}_s} d(l(v), l'(v))}{\sum_{v \in \mathcal{V}_s} \sqrt{(\Delta x_{max}(v))^2 + |y(l(v)) - y(l'(v))|^2}} \tag{4.15}$$

4.3.5 Relative Distance Metric

The *relative distance* metric [BT00] measures the average change in the distance between each pair of vertices between the two drawings $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}'(\mathcal{G})$ of the same graph $\mathcal{G}$. The idea is that preserving the relative ordering of the vertices is more important than preserving their absolute positions, specially if it is possible for the drawing (or a part of it) to transfer or rotate. In case of transferring or rotating, Euclidean distance produces a huge value of change although the mental map is not disturbed to the degree of the change of the computed value. For two drawings $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}'(\mathcal{G})$ of the same graph $\mathcal{G}$ with a matched point set $\mathcal{P}$, the relative distance is computed as:

$$\frac{1}{\mathcal{U} \times |\mathcal{P}| \times (|\mathcal{P}| - 1)} \sum_{1 \le i,j \le |\mathcal{P}|} \Big| d(p_i, p_j) - d(p'_i, p'_j) \Big| \tag{4.16}$$



(a) (b)

Figure 4.5: Cases of maximum distance a shared vertex v could move. The black circles represent the location of a shared vertex v in the current drawing $\mathcal{D}(\mathcal{H})$ where the grey ones represent the location of the same shared vertex v in the new drawing $\mathcal{D}'(\mathcal{H}')$.

where $\mathcal{U}$ is the drawing unit and $d(p_i, p_j)$ is the distance between the two points $p_i, p_j \in \mathcal{P}$ where p_i and p_j are the locations of the i^{th} and j^{th} in the drawing $\mathcal{D}$ and p'_i and p'_j are the locations of the same two points in the drawing $\mathcal{D}'$.

As with Euclidean distance, we could apply the relative distance only for the set of shared vertices of the two hierarchical graphs $\mathcal{H} = (\mathcal{V}_s \cup \mathcal{V}_a, \mathcal{E}_s \cup \mathcal{E}_a, \phi)$ and $\mathcal{H}' = (\mathcal{V}_s \cup \mathcal{V}'_a, \mathcal{E}_s \cup \mathcal{E}'_a, \phi')$. The relative distance metric μ_{RD} for the set of shared vertices V_s in the two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ is:

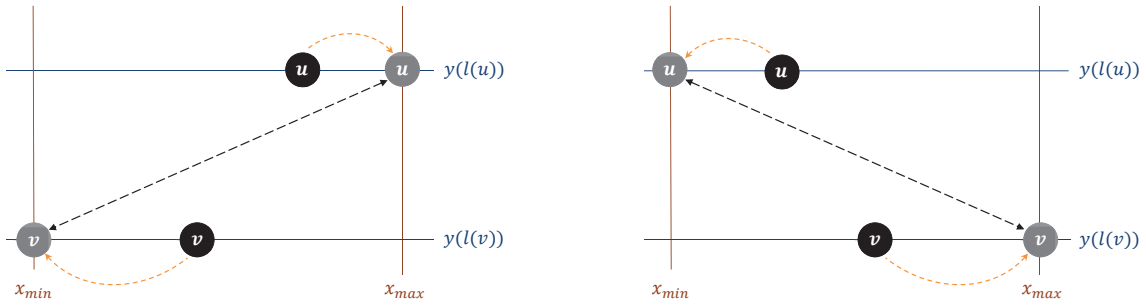
$$\frac{1}{\mathcal{U} \times |\mathcal{V}_s| \times (|\mathcal{V}_s| - 1)} \sum_{u,v \in \mathcal{V}_s} \left| d\left(l\left(u\right), l\left(v\right)\right) - d\left(l'\left(u\right), l'\left(v\right)\right) \right|$$

where $l(u)$ and $l(v)$ are the locations of the two shared vertices $u, v \in \mathcal{V}_s$ in the drawing $\mathcal{D}(\mathcal{H})$ respectively and $l'(u)$ and $l'(v)$ are the locations of the same two shared vertices u, v in the drawing $\mathcal{D}'(\mathcal{H}')$ respectively.

Also, as with the Euclidean distance metric, we have to normalize the relative distance metric value to be between 0 and 1. To do that, we have to compute the maximum distance could be found between two shared vertices u and v in the two drawings $\mathcal{D}$ and $\mathcal{D}'$. Assume x_{min} and x_{max} are the minimum and maximum x-coordinate in the drawing $\mathcal{D}$ respectively. The maximum horizontal distance u and v could found when one vertex is located on the left boundary and the other vertices is located on the right boundary. This means that the maximum horizontal distance between u and v is $x_{max} - x_{min}$. The maximum vertical distance Δy_{max} between u and v can be computed as:

$$\Delta y_{max}(u, v) = max\{y(l(v)), y(l'(v))\} - min\{y(l(u)), y(l'(u))\}$$

Figure 4.6 shows the maximum horizontal distance could be found between two shared vertices $u, v \in \mathcal{V}_s$. Now, the relative distance metric of the set of shared vertices in two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ of the two different hierarchical graphs $\mathcal{H}$ and $\mathcal{H}'$ respectively is:



(a) (b)

Figure 4.6: Cases of maximum horizontal move between two shared vertices u and v. Circles in black represent actual positions of the two shared vertices u and v in the current drawing $\mathcal{D}(\mathcal{H})$, where the circles in grey represents the horizontal positions of u and v in the new drawing $\mathcal{D}'(\mathcal{H}')$.

$$\mu_{RD}(\mathcal{D},\mathcal{D}') = \frac{\sum\limits_{u,v\in\mathcal{V}_s} |d(l(u),l(v)) - d(l'(u),l'(v))|}{\sum\limits_{u,v\in\mathcal{V}_s} \sqrt{(x_{max}-x_{min})^2 + (\Delta y_{max}(u,v))^2}} \tag{4.17}$$

4.3.6 Vertex Minimum Angle Metric

The *vertex minimum angle deviation* of a vertex $v \in \mathcal{V}$ in a drawing $\mathcal{D}(\mathcal{G})$ of a general graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is the ratio of the minimum angle of the vertex adjacent incident edges to the ideal angle of that vertex v, and computed as:

$$\left| \frac{\vartheta(v) - \theta_{min}(v)}{\vartheta(v)} \right|$$

where $\theta_{min}(v)$ is the actual minimum angle between the set of edges incident at vertex v, and $\vartheta(v)$ is the ideal (maximum) minimum angle at v. See the example given in Figure 4.7. If $d(v)$ is the degree of vertex v, the ideal angle $\vartheta(v)$ is computed as:

$$\vartheta(v) = \frac{360^\circ}{d(v)}$$

The *vertex minimum angle aesthetic metric* [Pur02] of an identical drawing $\mathcal{D}(\mathcal{G})$ of a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is the average of the vertices minimum angle deviation, as represented in Equation 4.18.

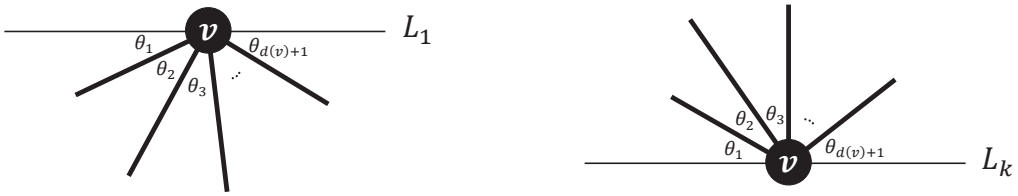
$$\frac{1}{|\mathcal{V}|} \sum_{v\in\mathcal{V}} \left| \frac{\vartheta(v) - \theta_{min}(v)}{\vartheta(v)} \right| \tag{4.18}$$

(a) (b)

Figure 4.7: Example of vertex minimum angle deviation. Drawing (a) has a minimum angle equals to the ideal angle ($\theta_{min} = \vartheta = 120°$) vertex v, where drawing (b) has a minimum angle $\theta_{min} = 45°$ and ideal angle $\vartheta = 120°$ at vertex v.

Now we formulate the vertex minimum angle aesthetic metric of two drawings of two different hierarchical graphs. Firstly, since any vertex $v \in \mathcal{V}_1$ have no incoming edges and any vertex $v \in V_k$ have no outing edges (see Figure 4.8), the ideal angle $\vartheta(v)$ of vertex $v \in \mathcal{V}_i$, $1 \leq i \leq k$ in the hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$ is modified to:

$$\vartheta(v) = \begin{cases} \dfrac{180°}{d(v)+1} & \text{if } v \in (\mathcal{V}_1 \cup \mathcal{V}_k); \\ \dfrac{360°}{d(v)} & \text{if } v \in \mathcal{V} \setminus (\mathcal{V}_1 \cup \mathcal{V}_k); \end{cases} \tag{4.19}$$



(a) Angles between incident edges of a first layer vertex $v \in \mathcal{V}_1$

(b) Angles between incident edges of a last layer vertex $v \in \mathcal{V}_k$

Figure 4.8: Angles between incident edges of a boundary layer vertex $v \in \mathcal{V}_1 \cup \mathcal{V}_k$ in a hierarchical graph $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$.

Then, for the two hierarchical graphs $\mathcal{H} = (\mathcal{V}_s \cup \mathcal{V}_a, \mathcal{E}_s \cup \mathcal{E}_a, \phi)$ and $\mathcal{H}' = (\mathcal{V}_s \cup \mathcal{V}'_a, \mathcal{E}_s \cup \mathcal{E}'_a, \phi')$, the change in the vertex minimum angle metric $\mu_{\mathcal{V}\vartheta}(\mathcal{D}, \mathcal{D}')$ of the two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ can be computed by applying our general framework in Equation 4.11 as:

$$\mu_{\mathcal{V}\vartheta}(\mathcal{D},\mathcal{D}') = \frac{\sum\limits_{v\in\mathcal{V}_s} |\theta_{min}(l(v)) - \theta_{min}(l'(v))| + \left| \sum\limits_{v\in\mathcal{V'}_a} \theta_{min}(l(v)) - \sum\limits_{v\in\mathcal{V}_a} \theta_{min}(l(v)) \right|}{\sum\limits_{v\in\mathcal{V}_s\cup\mathcal{V}_a} \vartheta(l(v))} \tag{4.20}$$

where $l(v)$ and $l'(v)$ are the locations of any shared vertex $v \in \mathcal{V}_s$ in both drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively, $\theta_{min}(l(v))$ is the minimum angle between edges incident on vertex v, and $\vartheta(l(v))$ is the ideal angle at vertex v as represented in Equation 4.19.

4.3.7 Vertex Width Metric

Vertices could be represented geometrically with different geometric shapes, e.g., rectangle, circle, ellipse, etc. Normally the vertex shape contains some text string to identify the vertex or determine some description of the vertex role in the graph. For example, in an organization chart, a vertex could have the employee name, position or a brief description of a job. In these case, it is possible to update the string of the text contained inside the vertex shape. In some cases the vertex shape could vary in both horizontal and vertical directions and consequently the width and/or the height of the vertex shape could be modified. In most of the cases (as we assume), the height of a vertex is kept fixed and just the width is permitted to be modified. So, the width of the vertex shape is considered as an important parameter in preserving the user's mental map.

Simply, a vertex width could be considered as the width of the rectangle that covers the vertex shape, or could be considered as the number of letters contained in the vertex string (in case of the vertex has text content). The average vertex width of a drawing $\mathcal{D}(\mathcal{G})$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is:

$$\frac{1}{|\mathcal{V}|} \sum_{v\in\mathcal{V}} w(sh(v)) \tag{4.21}$$

where $w(sh(v))$ is the width of the shape $sh(v)$ that represents any vertex $v \in \mathcal{V}$.

For the two hierarchical graphs $\mathcal{H} = (\mathcal{V}_s \cup \mathcal{V}_a, \mathcal{E}_s \cup \mathcal{E}_a, \phi)$ and $\mathcal{H}' = (\mathcal{V}_s \cup \mathcal{V}'_a, \mathcal{E}_s \cup \mathcal{E}'_a, \phi')$, the change in the vertex width metric $\mu_{\mathcal{VW}}$ of the two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ can be computed, as in Equation 4.22, by applying our general framework presented in Equation 4.11.

$$\mu_{\mathcal{VW}}(\mathcal{D},\mathcal{D}') = \frac{\sum\limits_{v\in\mathcal{V}_s} |w(sh(v)) - w(sh'(v))| + \left| \sum\limits_{v\in\mathcal{V'}_a} w(sh(v)) - \sum\limits_{v\in\mathcal{V}_a} w(sh(v)) \right|}{|\mathcal{V}| \times w_{max}} \tag{4.22}$$

where $sh(v)$ and $sh'(v)$ are the two shapes represent any shared vertex $v \in \mathcal{V}_s$ in both drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively, and w_{max} is the maximum vertex width in drawing $\mathcal{D}$, i.e., $w_{max} = \max\{w\,(sh\,(v))\}, v \in \mathcal{V}$.

4.3.8 Area Difference Metric

The area of a drawing is the area of the rectangle with that covers the drawing. The area difference metric considers the relative difference change of the area of the two drawings $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}'(\mathcal{G}')$. Simply, the area difference metric can be represented as:

$$\mu_{\mathcal{AR}}(\mathcal{D}, \mathcal{D}') = \left| 1 - \frac{Ar(\mathcal{D}')}{Ar(\mathcal{D})} \right| \tag{4.23}$$

where $Ar(\mathcal{D})$ and $Ar(\mathcal{D}')$ are the drawings of the two rectangles covers the two drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively.

4.3.9 Edge Orthogonality Metric

In a straight-line drawing of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *edge orthogonal deviation factor* $\delta(e)$ of an edge $e \in \mathcal{E}$ represents how far away from an orthogonal angle the edge straight line segment has deviated [Pur02]. It is computed as a proportion of the angular deviation of the line segment of edge e from the horizontal or vertical grid lines:

$$\delta(e) = \frac{\theta_{min}(e)}{45^\circ}, \quad \theta_{min}(e) = \min\left(\theta\left(e\right), \left|90^\circ - \theta\left(e\right)\right|, 180^\circ - \theta\left(e\right)\right)$$

where $\theta(e)$ is the positive angle between edge e and the x-axis, restricted to the range $0^\circ \leq \theta(e) \leq 180^\circ$ and $0 \leq \delta(e) \leq 1$.

For a drawing $\mathcal{D}(\mathcal{G})$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *edge orthogonality metric* is the average orthogonal deviation over all edge segments in $\mathcal{D}$ and is defined as in Equation 4.24.

$$\frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \delta(e) \tag{4.24}$$

Figure 4.9 represents a drawing of a graph with 6 edges where the orthogonal deviation of each edge is shown in the drawing.

Since the maximum orthogonal deviation angle for an edge segment in a graph drawing is 45°, the maximum edge orthogonal value for a drawing of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be computed as $45^\circ \times |\mathcal{E}|$. The change in the edge orthogonality metric $\mu_{E\delta}$ of the two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ of the two hierarchical graphs $\mathcal{H} = (\mathcal{V}_s \cup \mathcal{V}_a, \mathcal{E}_s \cup \mathcal{E}_a, \phi)$ and $\mathcal{H}' = (\mathcal{V}_s \cup \mathcal{V}'_a, \mathcal{E}_s \cup \mathcal{E}'_a, \phi')$ can be computed, as in Equation 4.25, by applying our general framework presented in Equation 4.12.

$$\mu_{\mathcal{E}\delta}(\mathcal{D}, \mathcal{D}') = \frac{\sum_{e \in \mathcal{E}_s} |\theta_{min}(s(e)) - \theta_{min}(s'(e))| + \left| \sum_{e \in \mathcal{E}'_a} \theta_{min}(s(e)) - \sum_{e \in \mathcal{E}_a} \theta_{min}(s(e)) \right|}{45^\circ \times |\mathcal{E}|} \tag{4.25}$$

where $s(e)$ and $s'(e)$ are the two line segments represent any shared edge $e \in \mathcal{E}_s$ in both drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively.

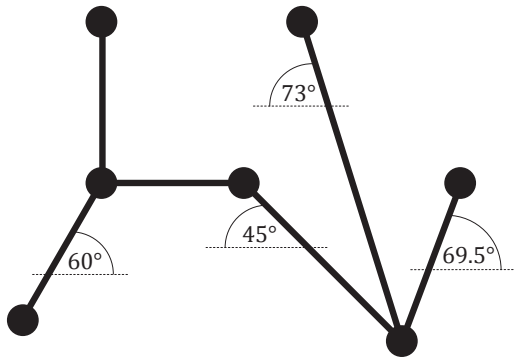


Figure 4.9: A graph drawing with 6 edges regarding edge orthogonality metric. The edge orthogonality metric of the graph drawing is computed as: $\mu_{\mathcal{E}\delta}(\mathcal{D}) = \frac{1}{6}\left(0 + 0 + \frac{30}{45} + \frac{45}{45} + \frac{17}{45} + \frac{20.5}{45}\right) = \frac{1}{6}\left(1 + 0.67 + 0.38 + 0.46\right) = 0.42$.

4.3.10 Edge Length Metric

The *edge length aesthetic metric* measures the average change in length of each edge in the first drawing $\mathcal{D}(\mathcal{G})$ and the second one $\mathcal{D}'(\mathcal{G})$ of the same graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The idea is based on the intuition that a great change in edge length makes the second drawing look so different. The length of an edge is the Euclidean distance between the edge source and target vertices locations. The edge length metric measures is represented as:

$$\frac{1}{|\mathcal{E}|}\sum_{e\in\mathcal{E}}\mathcal{L}(e) \tag{4.26}$$

where $\mathcal{L}(e)$ is the length of e.

For an edge $e = (u, v)$ in a hierarchical graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \phi)$ with drawing $\mathcal{D}(\mathcal{H})$, if x_{min} and m_{max} are the horizontal boundaries of $\mathcal{D}$, the maximum value of the length of $e \in \mathcal{E}$ is $\sqrt{|x_{max} - x_{min}|^2 + |y(l(u)) - y(l(v))|^2}$, where $l(u)$ and $l(v)$ are the locations of the two vertices u and v of the edge $e = (u, v)$. This maximum edge length is the same as the maximum distance between two shared vertices in the relative distance metric in Section 4.3.5.

By applying the proposed general framework presented in Equation 4.12, the edge length aesthetic metric for the two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ of the two hierarchical graphs $\mathcal{H} = (\mathcal{V}_s \cup \mathcal{V}_a, \mathcal{E}_s \cup \mathcal{E}_a, \phi)$ and $\mathcal{H}' = (\mathcal{V}_s \cup \mathcal{V}'_a, \mathcal{E}_s \cup \mathcal{E}'_a, \phi')$ is:

$$\mu_{\mathcal{E}\mathcal{L}}(\mathcal{D}, \mathcal{D}') = \frac{\sum\limits_{e\in\mathcal{E}_s}|\mathcal{L}(s(e)) - \mathcal{L}(s'(e))| + \left|\sum\limits_{e\in\mathcal{E}'_a}\mathcal{L}(s(e)) - \sum\limits_{e\in\mathcal{E}_a}\mathcal{L}(s(e))\right|}{\sum\limits_{e=(u,v)\in\mathcal{E}}\sqrt{|x_{max} - x_{min}|^2 + |y(l(u)) - y(l(v))|^2}} \tag{4.27}$$

where $s(e)$ and $s'(e)$ are the two line segments represent any shared edge $e \in \mathcal{E}_s$ in both drawings $\mathcal{D}$ and $\mathcal{D}'$ respectively.

4.3.11 Edge Crossing Metric

The importance of the number of edge crossings could be induced from the intuition: "*increasing the number of edge crossings in a graph drawing decreases the understandability of the graph*" [Pur97]. When calculating the number of crossings in a drawing, only pairwise edge intersections are considered. To produce a metric value of a drawing $\mathcal{D}$ between 0 and 1, the number of *actual* crossings C needs to be scaled against the number of *maximum possible* crossings C_{max}, i.e.:

$$\frac{C(\mathcal{D})}{C_{max}(\mathcal{D})} \tag{4.28}$$

For the purposes of the definition of a metric that can be universally applied to a graph drawing of any structure, a reasonable upper bound of the number of edge crossings is needed. Some forms for the upper bound of the number of crossings introduced in [SV97] and [PT00], but these forms were not so accurate [Pur02]. A reasonable approximation for the maximum possible crossings C_{max} in a graph drawing is defined by Purchase [Pur02] by subtracting the number of *impossible* crossings C_{imp} in that drawing from the number of *all* crossings if every edge crosses all other edges C_{all}, i.e.,

$$C_{max}(\mathcal{D}) = C_{all}(\mathcal{D}) - C_{imp}(\mathcal{D})$$

In a general (not-necessarily proper) hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$, an edge could connect two vertices belong to not necessary consecutive layers while no crossing could be produced by two edges share the same source vertex. So that, for a vertex v_j of order j in layer $\mathcal{V}_i$, with the outdegree $d^+(v_j)$, the maximum possible number of crossings $C_{all}(v_j)$, the number of impossible crossings $C_{imp}(v_j)$ and the number of maximum crossings $C_{max}(v_j)$ are:

$$\begin{aligned}
C_{all}(v_j) &= d^+(v_j) \times \left(\sum_{l=j+1}^{|\mathcal{V}_i|} d^+(v_l) + \sum_{p=i+1}^{k-1} \sum_{q=1}^{|\mathcal{V}_p|} d^+(v_q) \right) \\
C_{imp}(v_j) &= \frac{d^+(v_j) \times (d^+(v_j) - 1)}{2} \\
C_{max}(v_j) &= d^+(v_j) \times \left[\left(\sum_{l=j+1}^{|\mathcal{V}_i|} d^+(v_l) + \sum_{p=i+1}^{k-1} \sum_{q=1}^{|\mathcal{V}_p|} d^+(v_q) \right) - \frac{d^+(v) - 1}{2} \right]
\end{aligned}$$

Hence, the maximum possible number of crossings $C_{all}(\mathcal{D})$, the number of impossible crossings $C_{imp}(\mathcal{D})$ and the number of maximum crossings $C_{max}(\mathcal{D})$ of a drawing $\mathcal{D}$ of the hierarchical graph $\mathcal{H}$ are:

$$C_{all}(\mathcal{D}) = \sum_{i=1}^{k-1}\sum_{j=1}^{|\mathcal{V}_i|}\left[d^+(v_j) \times \left(\sum_{l=j+1}^{|\mathcal{V}_i|} d^+(v_l) + \sum_{p=i+1}^{k-1}\sum_{q=1}^{|\mathcal{V}_p|} d^+(v_q)\right)\right]$$

$$C_{imp}(\mathcal{D}) = \sum_{i=1}^{k-1}\sum_{j=1}^{|\mathcal{V}_i|} \frac{d^+(v_j)\,(d^+(v_j)-1)}{2}$$

Hence, the number of maximum possible edge crossings $C_{max}(\mathcal{D})$ is:

$$C_{max}(\mathcal{D}) = \sum_{i=1}^{k-1}\sum_{j=1}^{|\mathcal{V}_i|} d^+(v) \times \left[\left(\sum_{l=j+1}^{|\mathcal{V}_i|} d^+(v_l) + \sum_{p=i+1}^{k-1}\sum_{q=1}^{|\mathcal{V}_p|} d^+(v_q)\right) - \frac{d^+(v)-1}{2}\right] \tag{4.29}$$

The actual number of edge crossings C is therefore scaled against the possible number of crossings C_{max}, in order to normalize the edge crossing metric value between 0 and 1. Hence, the change in the edge crossing aesthetic metric $\mu_{\mathcal{E}C}$ of the two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ of the two hierarchical graphs $\mathcal{H}$ and $\mathcal{H}'$ respectively is:

$$\mu_{\mathcal{E}C}(\mathcal{D},\mathcal{D}') = \frac{|C(\mathcal{D}) - C(\mathcal{D}')|}{C_{max}(\mathcal{D})}$$

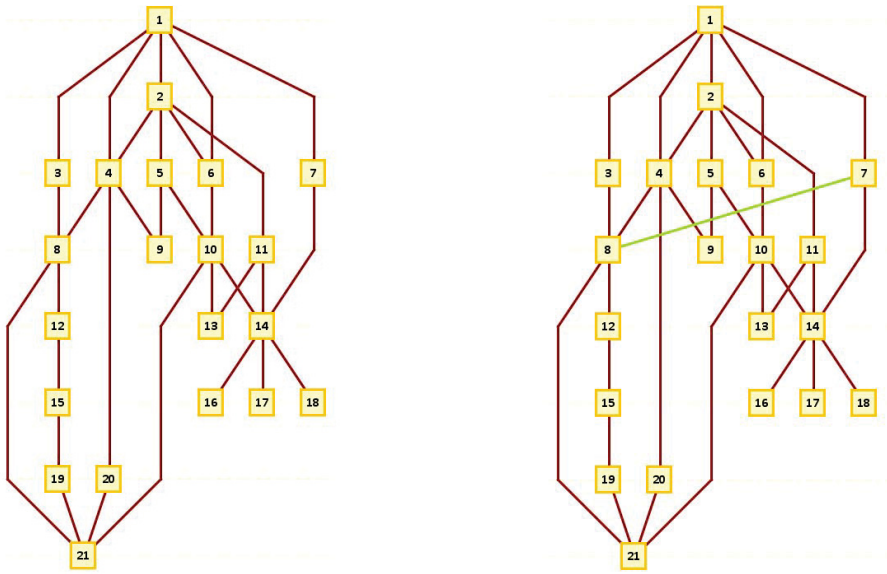
$$= \frac{|C(\mathcal{D}) - C(\mathcal{D}')|}{\sum_{i=1}^{k-1}\sum_{j=1}^{|\mathcal{V}_i|} d^+(v) \times \left[\left(\sum_{l=j+1}^{|\mathcal{V}_i|} d^+(v_l) + \sum_{p=i+1}^{k-1}\sum_{q=1}^{|\mathcal{V}_p|} d^+(v_q)\right) - \frac{d^+(v)-1}{2}\right]} \tag{4.30}$$

4.3.12 Examples

In this section, we introduce two examples. The first one considers computing the difference metrics for hierarchical graphs, introduced in Section 4.3. The second example shows the behaviour of the considered metrics in case of more than one drawing could be produced when executing an action on a hierarchical graph.

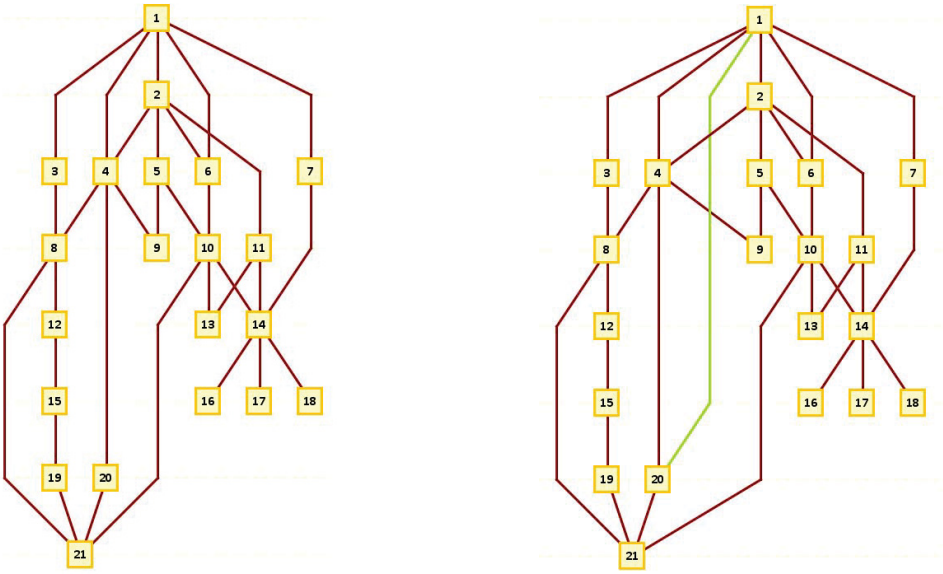
4.3.12.1 Example 1: Different Actions

In this example, we apply all the 13 possible actions come from the product in Section 4.3.1. The 13 graphs are shown in Figures 4.10-4.22. Each first subgraph (a) in all these 13 figures represents the current drawing $\mathcal{D}$ of a hierarchical graph with 21 vertices in 8 layers and 46 edges, where the second subgraphs (b) in these 13 figures represents the new drawing $\mathcal{D}'$ after executing an action. Also, Table 4.1 shows the computed values of the considered difference metrics for these 13 figures.



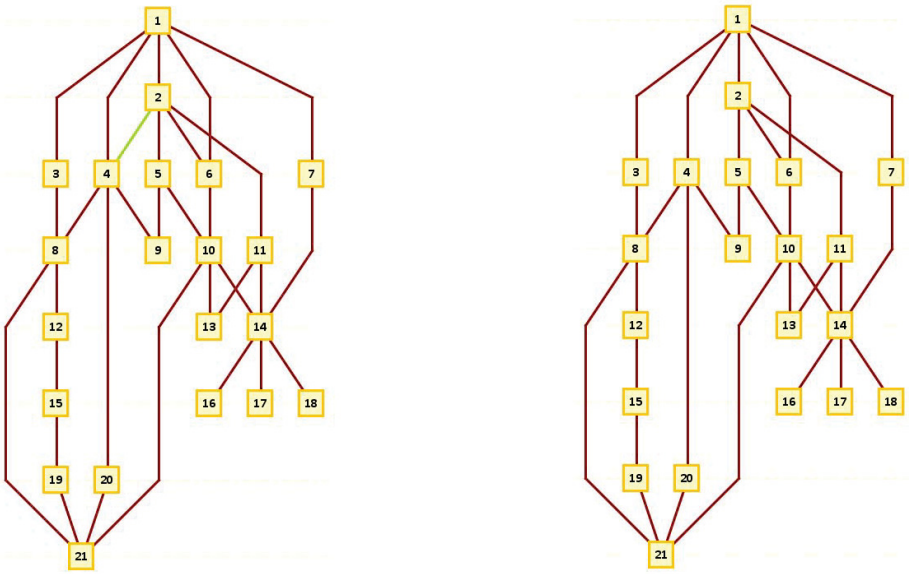
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.10: Example of inserting short edge action: inserting short edge (7,8) to connect the two vertices 7 and 8.



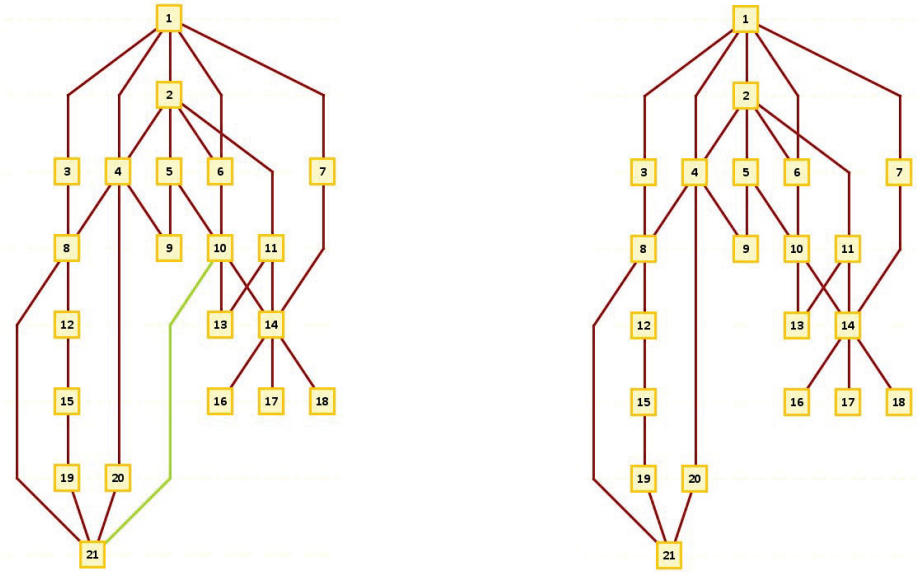
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.11: Example of inserting short edge action: inserting short edge (1,20) to connect the two vertices 1 and 20



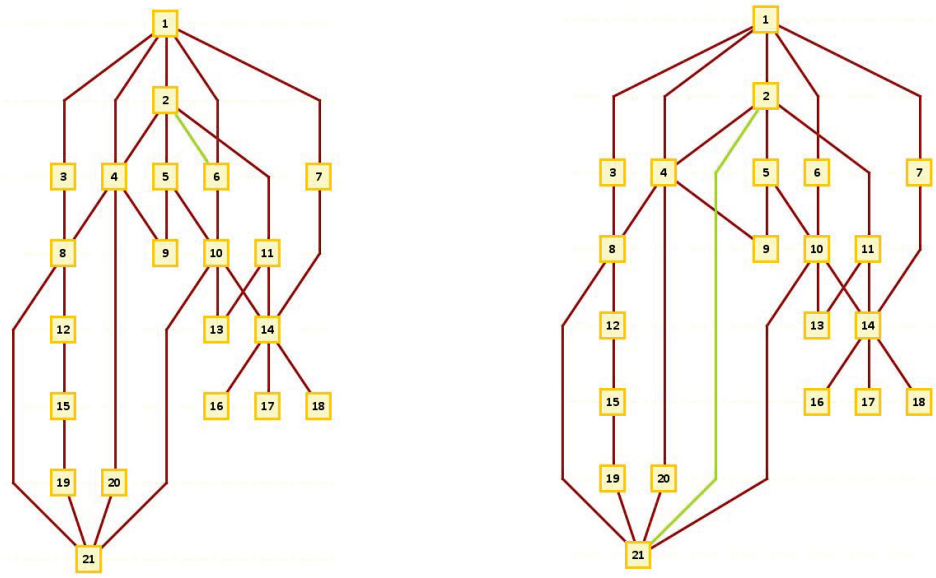
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.12: Example of deleting an existing short edge action: deleting short edge (2,4) that is connecting the two vertices 2 and 4.



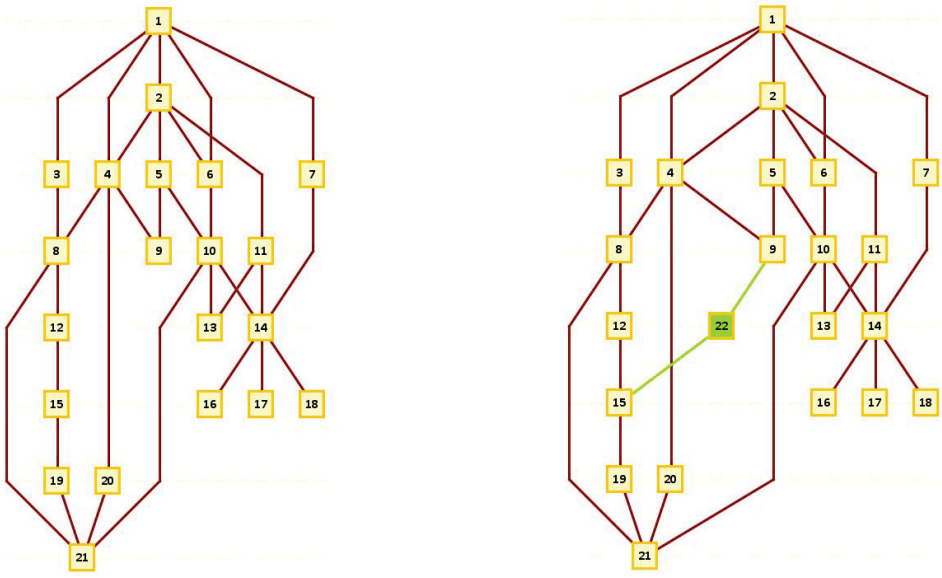
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.13: Example of deleting an existing long edge action: deleting long edge (10,21) that is connecting the two vertices 10 and 21.



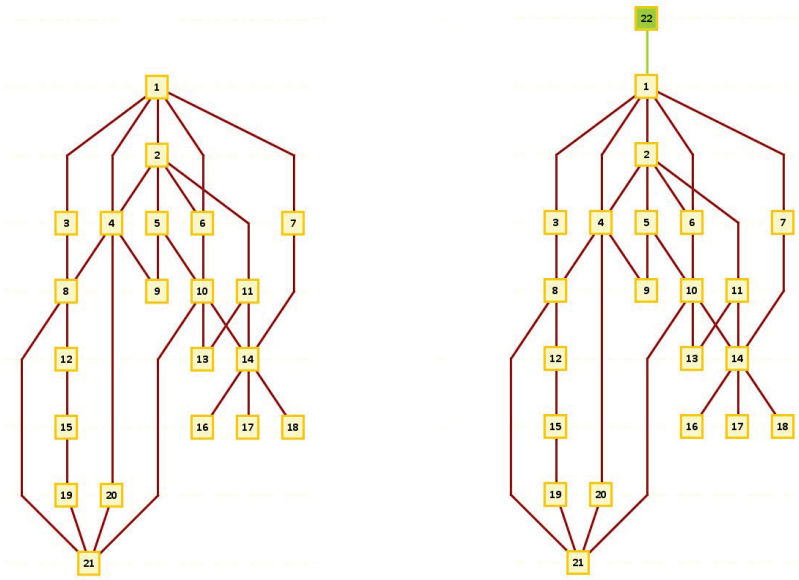
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.14: Example of modifying an existing edge action: modifying the short edge (2,4) that connects the two vertices 2 and 4 to be a long one (2,21) to connect the two vertices 2 and 21.



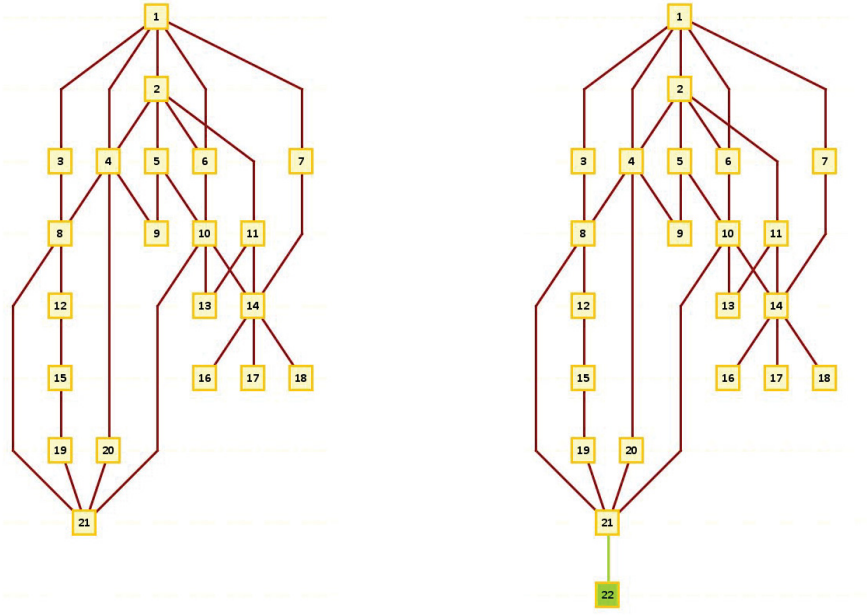
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.15: Example of inserting a new vertex to an existing layer action: inserting a new vertex 22 to the fifth layer to connect the two vertices 9 and 15 with the two new short edges (9,22) and (22,15).



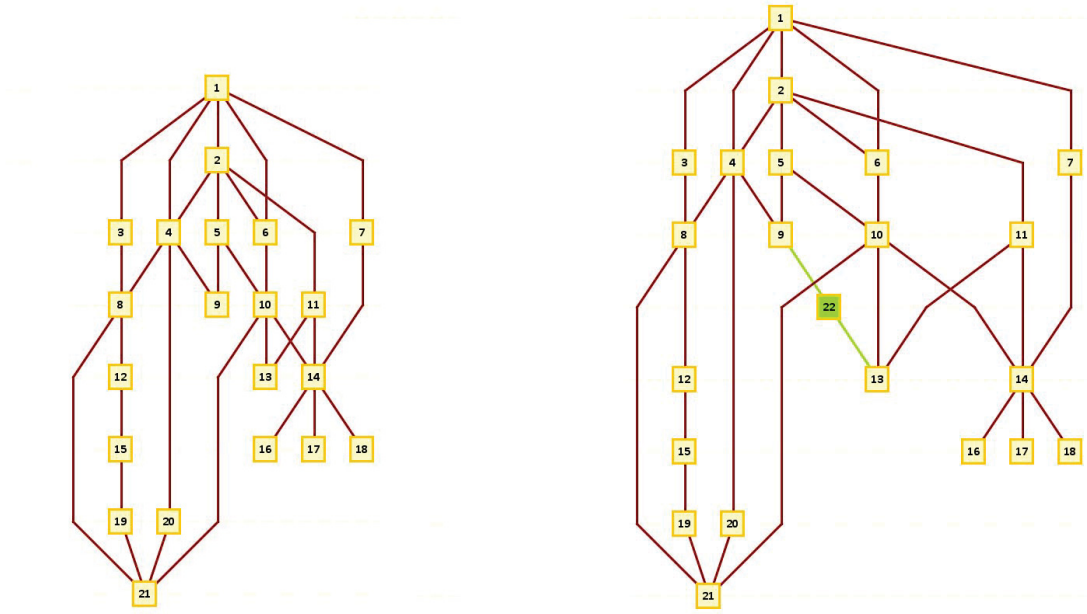
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.16: Example of inserting a new vertex in a new first layer action: inserting the new vertex 22 in a new first layer to connect vertex 1 with the new short edge (22,1).



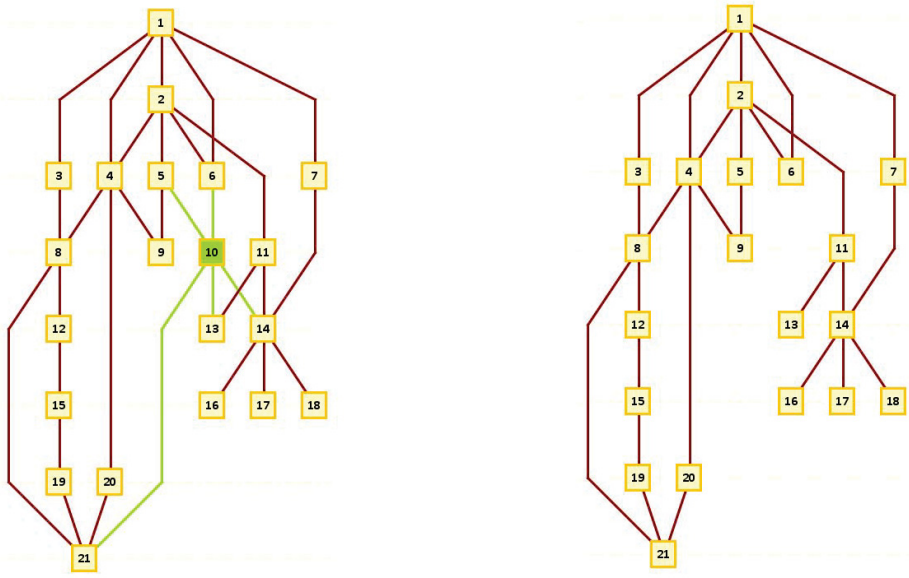
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.17: Example of inserting a new vertex in a new last layer action: inserting a new vertex 22 in a new last layer to connect the vertex 21 with a new short edges (21,22).



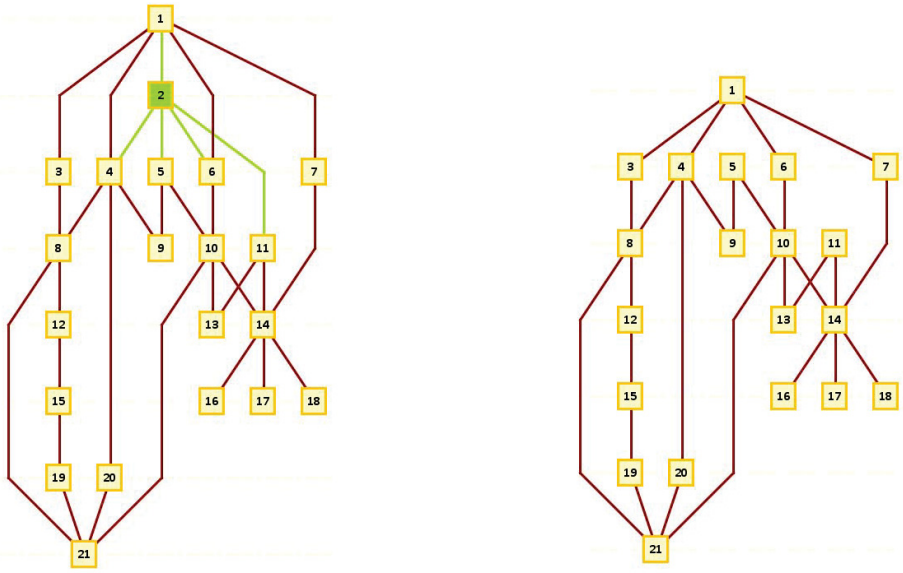
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.18: Example of inserting a new vertex in a new intermediate layer action: inserting a new vertex 22 in a new intermediate layer to connect the two vertices 9 and 13 with two new short edges (9,22) and (22,13).



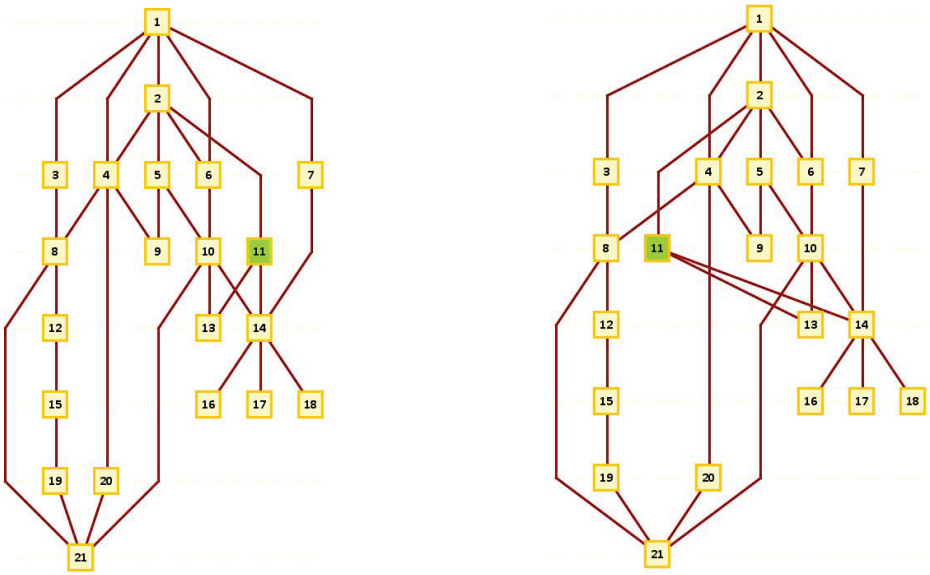
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.19: Example of deleting a vertex from its layer action: deleting vertex 2 and its adjacency edges, since vertex 11 is not the only non-dummy vertex in its layer, the layer will remain in the graph.



(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.20: Example of deleting a vertex and its layer action: deleting vertex 2 and its adjacency edges, since vertex 2 is the only non-dummy vertex in its layer, and the long edges connecting vertices in the first and third layers will be modified to short ones.

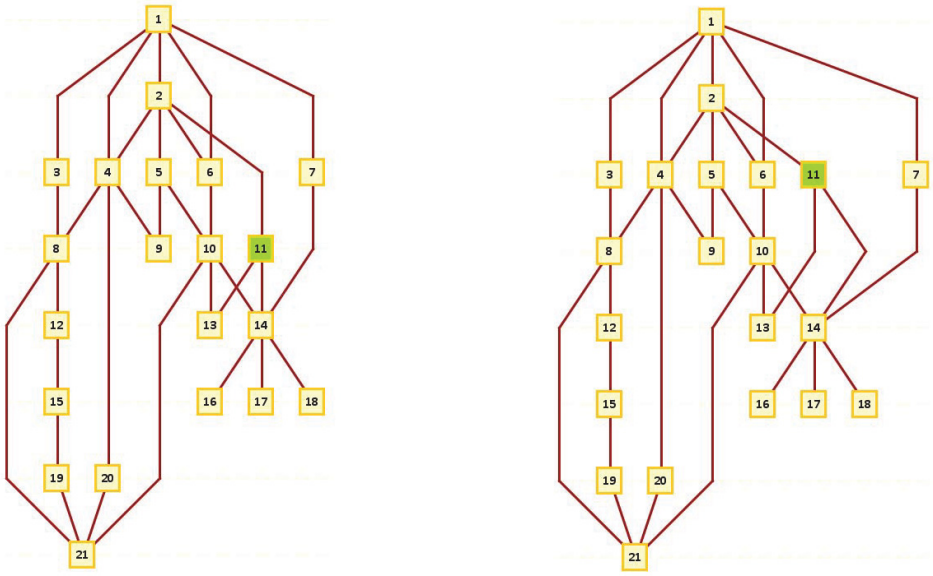


(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.21: Example of modifying vertex horizontal position in its layer action: moving vertex 11 from its original position to be after vertex 8.

Table 4.1: Difference aesthetic metrics values for the 13 graphs presented in Figures 4.10-4.22.

Action	Figure	Euclidean distance metrics $\mu_{\mathcal{ED}}$	Relative distance metrics $\mu_{\mathcal{RD}}$	Vertex minimum angle metric $\mu_{\mathcal{V}\vartheta}$	Edge orthogonality metric $\mu_{E\delta}$	Edge length metric $\mu_{\mathcal{EL}}$	Edge crossing metric $\mu_{\mathcal{EC}}$
insert a new short edge	4.10	0.0000	0.0000	0.0205	0.0013	0.0182	0.0147
insert a new long edge	4.11	0.1238	0.0700	0.1606	0.0214	0.0361	0.0049
delete an existing short edge	4.12	0.0000	0.0000	0.0000	0.0057	0.0045	0.0000
delete an existing long edge	4.13	0.0000	0.0000	0.0704	0.0098	0.0159	0.0000
modify an existing edge	4.14	0.1238	0.0700	0.1656	0.0141	0.0332	0.0024
insert a new vertex in an existing layer	4.15	0.1238	0.0700	0.0567	0.0189	0.0285	0.0024
insert a new vertex in a new first layer	4.16	0.0000	0.0000	0.0153	0.0138	0.0027	0.0000
insert a new vertex in a new last layer	4.17	0.0000	0.0000	0.0153	0.0138	0.0027	0.0000
insert a new vertex in a new intermediate layer	4.18	0.2543	0.2215	0.3091	0.0256	0.0771	0.0024
delete a existing vertex from its layer	4.19	0.0000	0.0000	0.1092	0.0211	0.0303	0.0024
delete an existing vertex and its layer	4.20	0.0056	0.0066	0.1354	0.0450	0.0355	0.0024
modify vertex horizontal position in its layer	4.21	0.1269	0.0859	0.0377	0.0182	0.0362	0.0122
modify vertex layer	4.22	0.0198	0.0231	0.0250	0.0144	0.0130	0.0000



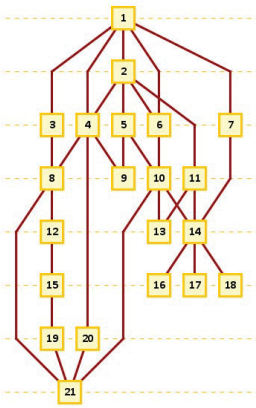
(a) Current drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 4.22: Example modifying vertex layer action: moving vertex 11 from its layer one layer up. This can be done since all its incoming edges to vertex 11 are long edges, consequently updating its adjacency edges is required.

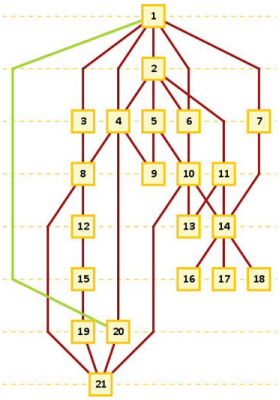
4.3.12.2 Example 2: Different Possible Drawings

Figure 4.23(a) represents a hierarchical graph with 21 vertices in 8 layers ordered from top to bottom and 46 edges. The action is to insert a new long edge (1,20) to connect the two vertices 1 and 20. There are 8 different possibilities to execute this action depending on where to insert the edge, i.e. the x-coordinates of the dummy vertices of the edge keeping the new long edge as straight as possible. There are 8 possibilities for inserting the long edge starting with inserting it in the far-left-hand side starting from the left boundary of the drawing and going inside the drawing till inserting it on the far-right-hand side. The 8 different possibilities are presented in Figure 4.23. The chart in Figure 4.24 represents the curves of the considered difference metrics introduced in Section 4.3 to compute the difference between the current drawing in Figure 4.23(a) and each new drawing from the 8 possibilities in Figures 4.23(b)-4.23(i).

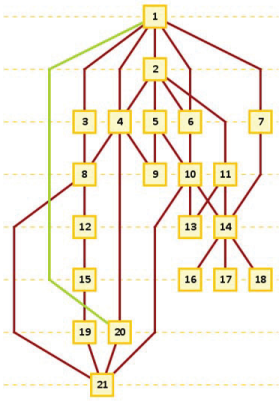
From this chart 4.24 we could observe that "*inserting new component as outside the graph drawing as possible leads to lower change in most of the considered metrics values.*" This observation could be easily captured from the chart since, for most of the metrics (except edge length metric $\mu_{\mathcal{EL}}$), the minimum value is in the first and final possible drawing, where the maximum value is in the middle possible drawing. This completely reflects that the change between both drawings is small as when a large set of the vertices and edges keep the same geometric values.



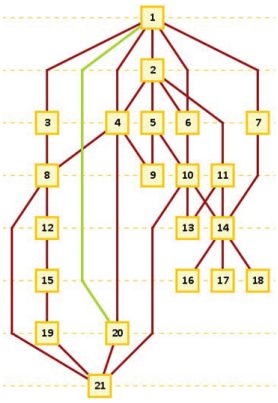
(a) original drawing $\mathcal{D}$ before executing any actions



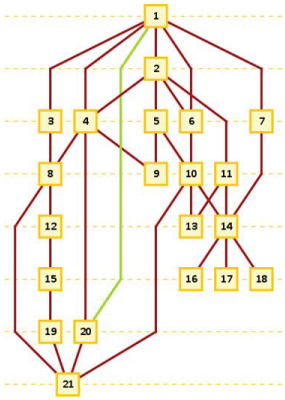
(b) New drawing $\mathcal{D}'$ (Possibility 1)



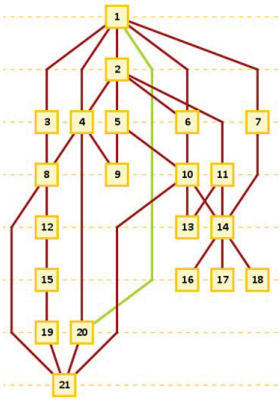
(c) New drawing $\mathcal{D}'$ (Possibility 2)



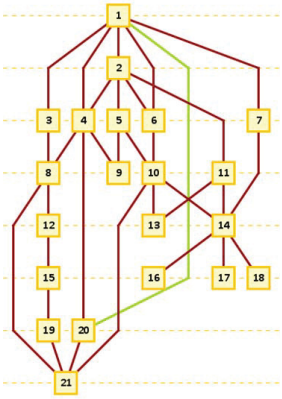
(d) New drawing $\mathcal{D}'$ (Possibility 3)



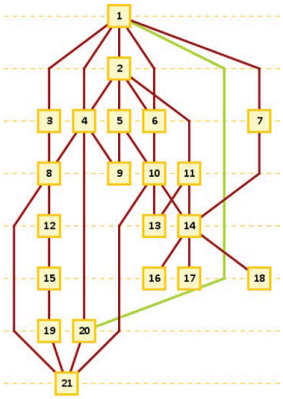
(e) New drawing $\mathcal{D}'$ (Possibility 4)



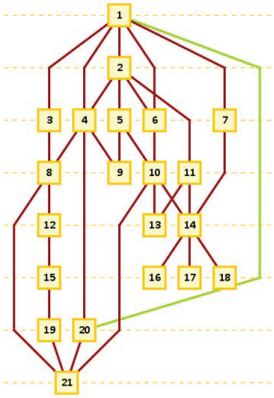
(f) New drawing $\mathcal{D}'$ (Possibility 5)



(g) New drawing $\mathcal{D}'$ (Possibility 6)



(h) New drawing $\mathcal{D}'$ (Possibility 7)



(i) New drawing $\mathcal{D}'$ (Possibility 8)

Figure 4.23: Current hierarchical graph drawing $\mathcal{D}$ in (a) and different 8 possibilities (b)-(i) for inserting new long edge (1,20).

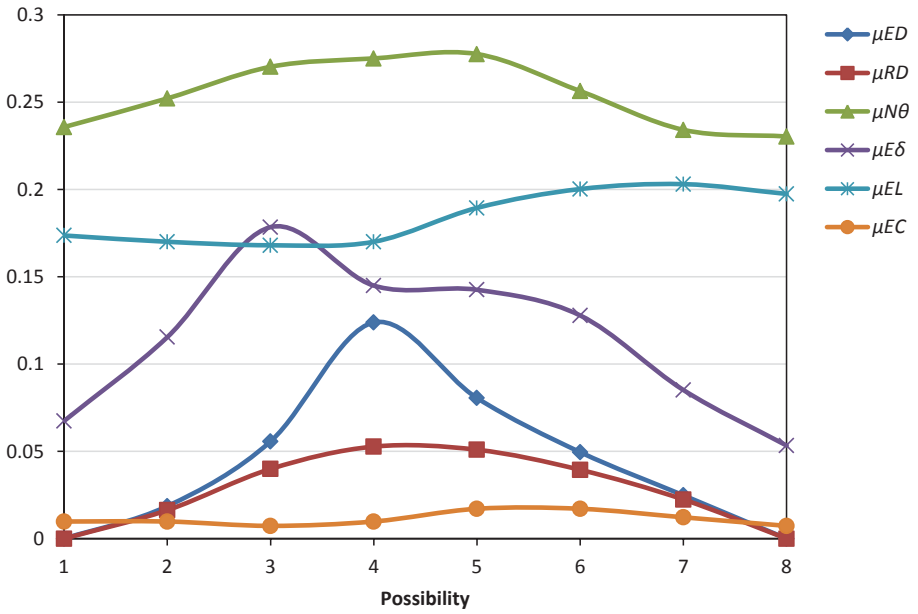


Figure 4.24: Difference metrics values for each of the 8 different possible drawings presented in Figure 4.23(b)-4.23(i) with the current drawing in Figure 4.23(a).

4.4 Crossing Minimization in Dynamic Hierarchical Graph Drawing

The preservation of structure from the previous graph drawing is found to be far superior in helping the users to re-orient themselves in a new view. It has been mentioned in preserving the mental user's map in Section 4.2, the user has to spend some effort in order to refamiliarize himself/herself with the new drawing after executing an action to the current drawing. The value/mass of the effort of user refamiliarity depends actually on how much the executed change has made the new drawing different from the current one. In other words, the lower difference between the new and the current drawings, the lower needed effort.

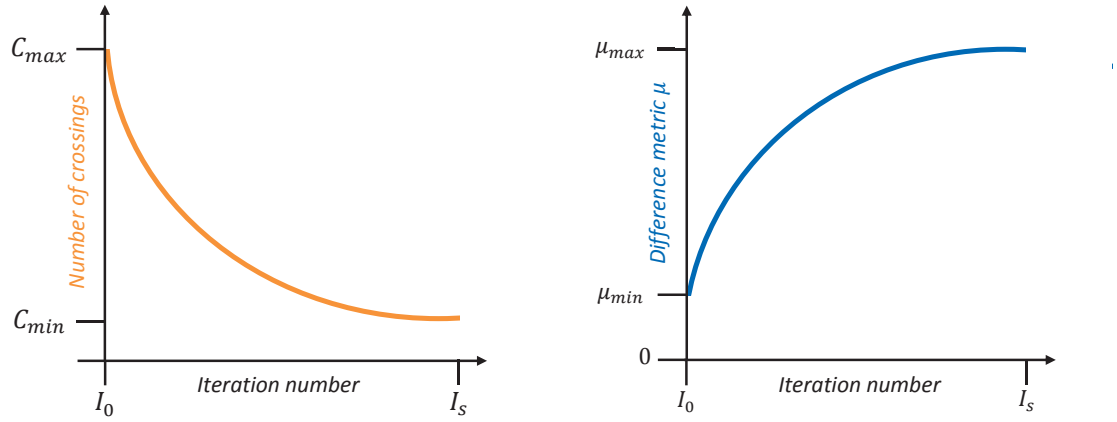
Many graph drawing algorithms are *deterministic*, i.e. they take a graph as input and generate a "single" drawing as its output based on a set of rules. On the other hand, *non-deterministic* algorithms will iteratively perturb the drawing of a graph until it reaches some heuristically defined quiescent state or a termination condition is reached. Actually, crossing minimization is solved using a non-deterministic algorithm. Non-deterministic drawing approaches, like the *spring layout* [Ead84], are generally used to find optimal layouts for a static data set. The important characteristic of dynamic graph drawings on the other hand is that the underlying data of the graph is changing, which means the algorithm must be able to adapt to these changes, but it is not a requirement that the drawing be done iteratively. This is not to say that the two cannot go together. We wish to draw a distinction between dynamic graph drawing and crossing minimization aesthetic criterion.

Almost of the crossing minimization algorithms, as they are non-deterministic, work heuristically and iteratively produce a drawing in each round till the convergence, and normally the final drawing has a minimum number of edge crossings. However, any drawing produced in an iteration of a crossings minimization algorithm in a dynamic graph drawing environment has two parameters: the number of crossings, and the mass of change between that drawing and the current one according to some difference metric. A considered question here is: from the finite drawings produced in every algorithm iteration, which drawing has a minimum number of edge crossings and in the same time has a minimum value of difference change (from the original drawing)?

The problem of minimizing edge crossings in dynamic graph drawing could be defined as:

> *suppose $\mathcal{D}(\mathcal{H})$ is the current drawing of a hierarchical graph $\mathcal{H}$ with a crossing number $C(\mathcal{D})$, when executing an action $\mathcal{A}$ on $\mathcal{D}(\mathcal{H})$ with restricting the changes of the current drawing $\mathcal{D}$, an initial new drawing $\mathcal{D'}_0(\mathcal{H'})$ of the new graph $\mathcal{H'}$ is produced. By transferring the drawing $\mathcal{D'}_0(\mathcal{H'})$ to an edge crossing minimization algorithm, a set of s drawings $\mathcal{D'}_1, \mathcal{D'}_2, \cdots, \mathcal{D'}_s$ are produced such that drawing $\mathcal{D'}_i$ is produced in iteration I_i. The objective is to find the drawing $\mathcal{D'}_i$, $0 \leq i \leq s$, such that $C(\mathcal{D'}_i)$ is minimized and $\mu(\mathcal{D}, \mathcal{D'}_i)$ is minimized.*

Consider the previous definition of the problem of minimizing edge crossings in dynamic graph drawing. Assume that C_{max} and C_{min} are the minimum and maximum number of crossings over all the s drawings produced in the s iterations when applying some crossing minimization algorithm to the new hierarchical graph $\mathcal{H'}$. Since the number of crossings is smoothly reduced through the algorithm flow over the s iterations from the initial iteration drawing till the final iteration one, the general behaviour of a crossing minimization algorithm could be similar to the chart in Figure 4.25(a).



(a) General behaviour of a crossing minimization algorithm through the $s+2$ iterations $I_0, I_1, \cdots, I_s$

(b) General behaviour of a difference metric μ through the $s+2$ iterations $I_0, I_1, \cdots, I_s$

Figure 4.25: Behaviours of crossing minimization algorithms and a difference metric μ.

Furthermore, suppose that some difference metric μ is used in measuring the difference between the current drawing $\mathcal{D}$ and the new $s+1$ drawings $\mathcal{D}'_0, \mathcal{D}'_1, \cdots, \mathcal{D}'_s$. Assume that $\mu_{max} = \max\{\mu_i\}$ and $\mu_{min} = \min\{\mu_i\}$, $\mu_i = \mu(\mathcal{D}, \mathcal{D}'_i)$, $0 \leq i \leq s$, are the maximum and minimum metric values, respectively. Experimentally, after applying the proposed difference metrics for hierarchical graphs (introduced in Section 4.3) to many graphs, we could say that the general behaviour of most of the difference metric μ is similar to the chart in Figure 4.25(b).

In order to compare the two curves of the crossings number and the difference metric values of each iteration, we have to scale the values of both curves to have the same minimum and maximum value. In order to do this, we divide each of the curve iteration values C_i and μ_i by the difference between its minimum and maximum values, respectively. So, the scaled value $\bar{C}_i$ of the number of crossings C_i of iteration i, and the scaled value $\bar{\mu}_i$ difference metric value μ_i of the same iteration i are:

$$\bar{C}_i = \frac{C_i}{C_{max} - C_{min}} \quad \text{and} \quad \bar{\mu}_i = \frac{\mu_i}{\mu_{max} - \mu_{min}}$$

The chart in Figure 4.25 represents the combination of both curves of the crossings number and the difference metric values such that all these values are normalized between 0 and 1.

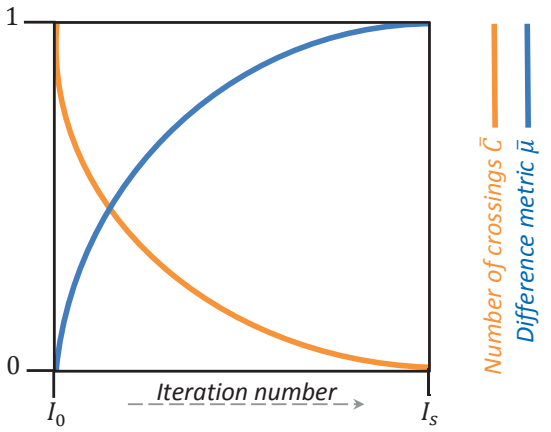


Figure 4.26: Behaviours of normalized values of crossing minimization algorithms and a difference metric.

An example of this problem is given in Figures 4.27 and 4.28. The drawing given in Figure 4.27 represents the current drawing $\mathcal{D}$ before executing any actions. The action to be executed is inserting a new vertex (number 13) with new 5 new short edges (9,13), (13,14), (13,15), (13,16) and (13,17). We insert vertex 13 with the new edges is done initially by keeping all the vertices coordinates and edge routing in drawing 4.28(a) as the same as in drawing 4.27. Now, we apply the Efficient Barycenter algorithm (Algorithm 3.12) to the new initial drawing 4.28(a). The algorithm takes 6 iterations till convergence. For each of the algorithm 6 iterations, the corresponding drawing is introduced in the 6 drawings 4.28(a) to 4.28(f). The question now is which one of these 6 drawings should be selected as the solution of crossing minimization problem in dynamic drawing scenario.

An ideal solution to this problem is to define a set of aesthetic qualities in advance. For example, the selected drawing $\mathcal{D}'$ should have ate least 75% similarity with the current drawing $\mathcal{D}$, at most 10% edge bendiness, at most 5% edge crossinginess, at least 65% edge orthogonality, etc. Another way to define these aesthetic qualities could be completely depend on a comparison between the two drawings $\mathcal{D}$ and $\mathcal{D}'$ such as: the difference between $\mathcal{D}$ and $\mathcal{D}'$ should be at most 5% edge crossing metric, at most 10% edge length metric, at most 7% edge orthogonality metric, at least 15% vertex minimum angle metric, etc.

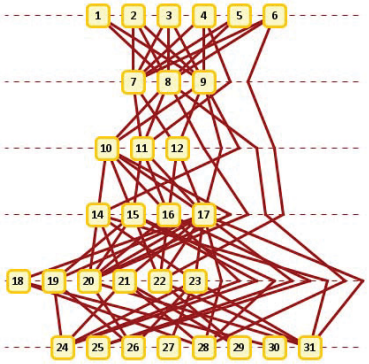


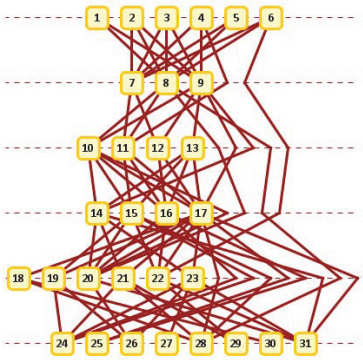
Figure 4.27: A hierarchical graph drawing.

4.5 Summary

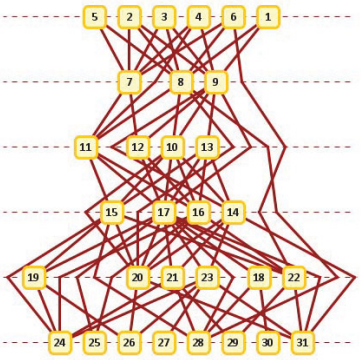
In many applications, graphs are dynamic where changes are executed to a graph in order to reflect the evolution of the system behaviour represented by that graph. In dynamic graph drawing scenarios, users have to spend time and effort on refamiliarizing themselves with the new drawn graphs and here the user's *mental maps* have to be preserved. Leaving the action to be executes freely and using some geometric aesthetic difference metrics in order to compute the change between the old and the new drawing is the recommended approach for preserving the mental map.

Although the metrics presented in [Pur02] are applicable to drawings of any graph of any structure or size, enabling quantitative comparisons between drawings of different graphs, the two metric values $\mu(\mathcal{D})$ and $\mu(\mathcal{D}')$ of the two identical drawings $\mathcal{D}$ and $\mathcal{D}'$ does not "precisely" reflect the change between these two drawings. While Bridgeman and Tamassia [BT00] defined some metrics associated with graph drawings, their concern is with the measurement of differences between two drawings of the same graph in a dynamic environment.

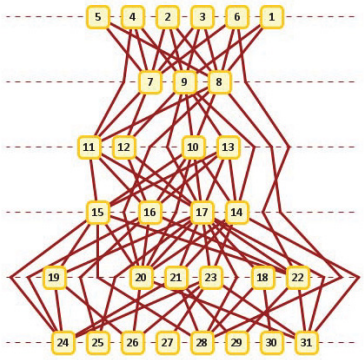
In this chapter, we introduced a general framework for defining and validating geometric aesthetic metrics to measure the difference between two drawings of two different hierarchical graphs when a change is executed. We have applied this proposed framework to some existing metrics that have been used in different drawing styles (like orthogonal drawings). The proposed framework could be applied to any hierarchical graph of any size and could also be extended to be applied to any graph type of any size.



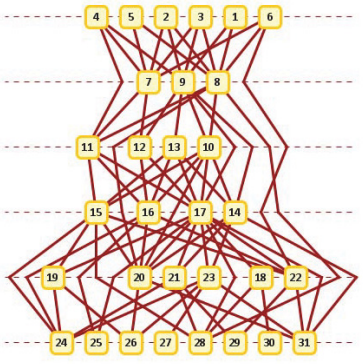
(a) New drawing $\mathcal{D}'$: Possibility 0



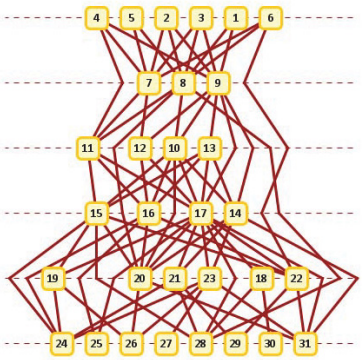
(b) New drawing $\mathcal{D}'$: Possibility 1



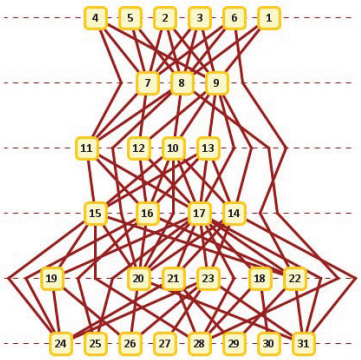
(c) New drawing $\mathcal{D}'$: Possibility 2



(d) New drawing $\mathcal{D}'$: Possibility 3

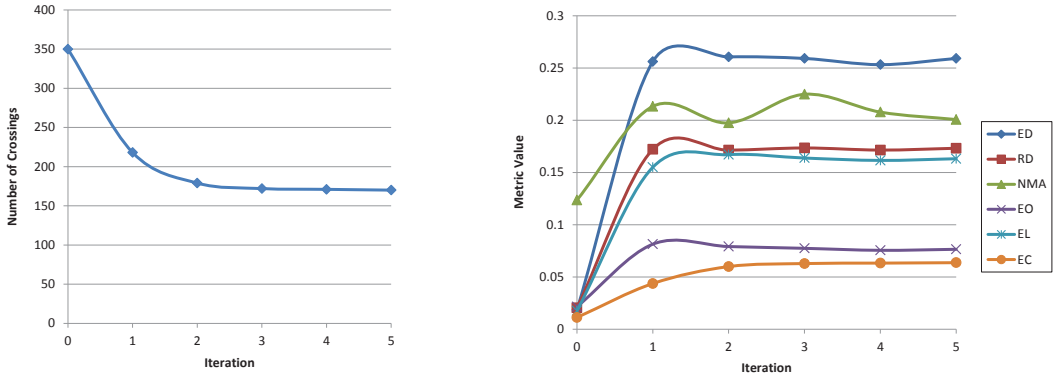


(e) New drawing $\mathcal{D}'$: Possibility 4



(f) New drawing $\mathcal{D}'$: Possibility 5

Figure 4.28: 6 new possible drawings of the 6 iterations produced using the Efficient Barycenter algorithm (Algorithm 3.12) for the executing action: inserting a new vertex 13 with new 5 short edges (9,13), (13,14), (13,15), (13,16) and (13,17)" to the drawing in Figure 4.27.



(a) (b)

Figure 4.29: (a) Crossing numbers of the 6 drawings produced in Figure 4.28 and (b) difference metrics values produced when computing the difference between each of the 6 drawings in Figure 4.28 and the drawing in Figure 4.27.

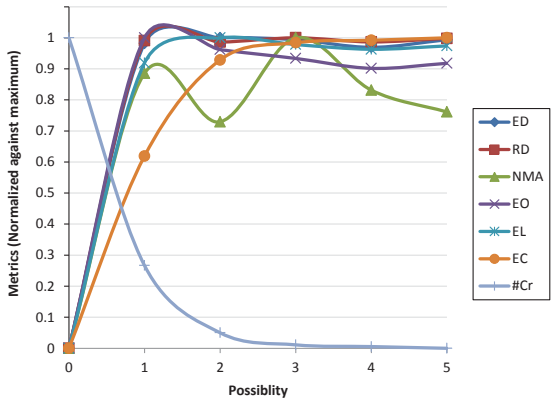


Figure 4.30: Values of the difference metrics and the number of crossings (scaled between 0 and 1) for the 6 possible drawings presented in Figure 4.28 compared with the original drawing presented in Figure 4.27.

The proposed framework depends on distinguishing between the graph components (vertices and edges) by identifying which one is strongly related to the action and which one not. The components related to the action, named action components, are those inserted into or deleted from the old graph. The rest components of the graph, named shared components, are those which are found in both graphs. We formulate a general expression of a difference metric depending on both metric values of the action and shared components. We close the chapter with an introduction to the problem of crossing minimization in dynamic hierarchical graph drawing.

5

A User Study in Difference Metrics for Hierarchical Graphs

Every beauty which is seen here by persons of perception resembles more than anything else that celestial source from which we all are come.[1]

In this chapter we introduce the results of an experimental user study in validating the proposed difference metrics of hierarchical graphs that are presented in Section 4.3.

5.1 Introduction

A great deal of research has been carried out on the algorithmic side of the graph drawing problem to produce an aesthetically pleasing drawing. While algorithm designers have developed a number of drawing criteria for what makes an effective layout, such as minimizing edge crossings or maximizing the angles of incidence of edges where they connect to vertices, the problems of optimizing such aesthetic criteria are computationally very challenging and so heuristics have to be adopted to achieve approximate solutions [DLF+09]. Further, improving the drawing with respect to a specific drawing criterion may require a trade-off with respect to one or more of the others. For example, algorithms exist to lay out a planar graph (with no crossings) but they do so at the expense of very poor angular resolution. For these reasons, algorithm design must be extended by studies of the human factors in a readable layout to decide which heuristics are most important for layout optimization.

[1] Michelangelo

The need for measuring the similarity or dissimilarity between two drawings of a dynamic graph arises in many problems. For example, in iterative graph drawing, the graph being drawn changes over time and hence, it is important to preserve the user's mental map. Furthermore, similarities between drawings can be used as a basis for indexing and retrieval. In character and handwriting recognition, a written character may be transformed into a graph drawing and then compared to a database of characters in order to find the closest match.

User studies offer a statistical method to measure the performance of a visualization. The reasons to pursue user studies are abundant, particularly when evaluating the strengths and weaknesses of different visualization techniques. Studies can validate whether a new visualization technique is useful in a practical sense, according to some objective criteria, for a specific task. User studies can objectively establish which method is most appropriate for a given situation. While user studies are an important tool for visualization design, they are not the appropriate choice in every situation. Experiments do not always work as expected and other techniques (like mathematical or statistical techniques) may be more convenient.

A more fundamental goal of conducting user studies is to seek insight into why a particular technique is effective. This can guide future efforts to improve existing techniques. We want to understand what types of tasks and conditions yield high-quality results for a particular method. This knowledge is critical because different analysis tasks require different visualization techniques.

A final use for studies in visualization is to show that an abstract theory applies under certain practical conditions. For example, results from a psychophysics or computer viewpoint may not extend to a visualization environment. We can run user studies to test this hypothesis. Results can show when the theories hold and how they need to be modified to function correctly for real-world data and tasks [SI10,KHI+03]. A good starting point in any study is the scientific or visual design question to be examined. This drives the process of experimental design. A poorly designed experiment will yield results of only limited value.

The chapter is organized into 3 parts from which this is the first one. Section 5.2 represents the design of the user study including the participants and their characteristics, the questions that the users have been asked, and the procedure of presenting and answering the questions. Section 5.3 contains the main results we derived from the users, answers besides those produced by the difference metrics.

5.2 Study Design

This study focuses on similarity measures for different drawings of a dynamic hierarchical graph. The focus here is to validate the mathematical formulations of difference metrics of dynamic hierarchical graphs presented in Section 4.3.

The graphs used were generated from a base set of 50 small graphs with 10-20 vertices each, randomly generated using the General Hierarchical Graph Generator (Algorithm 3.14). Forty modified drawings were created by executing the 13 possible actions of changing a hierarchical graph, as presented in Section 4.3. Depending on the executed action, it is possible to have one or more new drawings.

The experiment consisted of two parts, to address the evaluation criteria. In all cases, the user was asked to quantify the degree of similarity between two drawings of a dynamic hierarchical graph, and the user has to sort a set of more than one possible new drawing, after executing an action on a hierarchical graph, according to the similarity to the original drawing.

5.2.1 Participants

We recruited 20 participants (17 males and 3 females) representing the members of the Efficient Algorithms research group of the Institute of Applied Informatics and Formal Description Methods AIFB at the Karlsruhe Institute of Technology KIT. All of the participants have some information about graphs and their representations but no specialized information about graph drawing. The user study has been done during the *annual Dagstuhl Klausurtagung "LST Schmeck" 2011*, held 16-18.11.2011.

5.2.2 Tasks

Our primary research task is formulated in the following question:

> *Given the two drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ of the two hierarchical graphs $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$ and $\mathcal{H}' = (\mathcal{V}'_1, \mathcal{V}'_2, \cdots, \mathcal{V}'_{k'}; \mathcal{E}')$ respectively, such that $\mathcal{H}'$ is produced by executing an action $\mathcal{A}$ on the graph $\mathcal{H}$.*
> ***How much (in percent) is drawing $\mathcal{D}$ similar to drawing $\mathcal{D}'$?***
> The users are given the two drawings and are informed about the executed action.

The second task question is presented as follows:

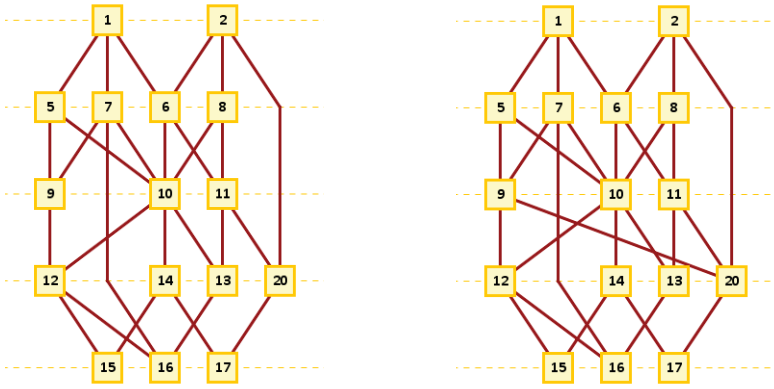
> *Given a drawing $\mathcal{D}(\mathcal{H})$ of a hierarchical graph $\mathcal{H} = (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_k; \mathcal{E})$ and a finite number s of different possible drawings $\mathcal{D}'_1(\mathcal{H}'), \mathcal{D}'_2(\mathcal{H}'), \cdots, \mathcal{D}'_s(\mathcal{H}')$ of another hierarchical graph $\mathcal{H}' = (\mathcal{V}'_1, \mathcal{V}'_2, \cdots, \mathcal{V}'_{k'}; \mathcal{E}')$ which is produced by executing an Action $\mathcal{A}$ on the graph $\mathcal{H}$. The drawings $\mathcal{D}'_1, \mathcal{D}'_2, \cdots, \mathcal{D}'_s$ are different possible drawings of $\mathcal{H}'$.*
> ***Order the s drawings $\mathcal{D}'_1, \mathcal{D}'_2, \cdots, \mathcal{D}'_s$ according to the degree of similarity to the original drawing $\mathcal{D}$.***
> *The users are given the $s+1$ drawings and are informed about the executed action.*

5.2.3 Procedure

A *PDF* file containing all the drawings of both questions is sent to each participant. Furthermore, each participant is given a paper with two tables, one for each question. In the first table, the participant is putting his evaluation percentage to the degree of similarity between each two of the given drawings. In the second table, for each graph, the participant inputs his ordering of the possible drawings with respect to the degree of similarity to the original drawing.

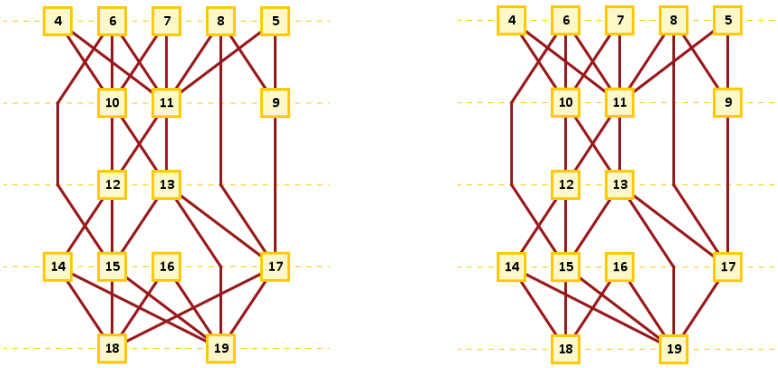
For task 1, the examples given to the users are shown in Figures 5.1-5.13. For the examples with more than one possible drawing, the users are given the original drawing and each of the possible drawings separately. For example, in Figure 5.3 there are 5 possible new drawings for inserting the long edge, hence the users are given 5 samples each having two drawings, which are the original drawing and one of the possible 5 new drawings. This technique gives the users 36 examples for all the possible new drawings in these figures.

For task 2, the examples given to the users are shown in Figures 5.3, 5.5, 5.6, and 5.10-5.13.



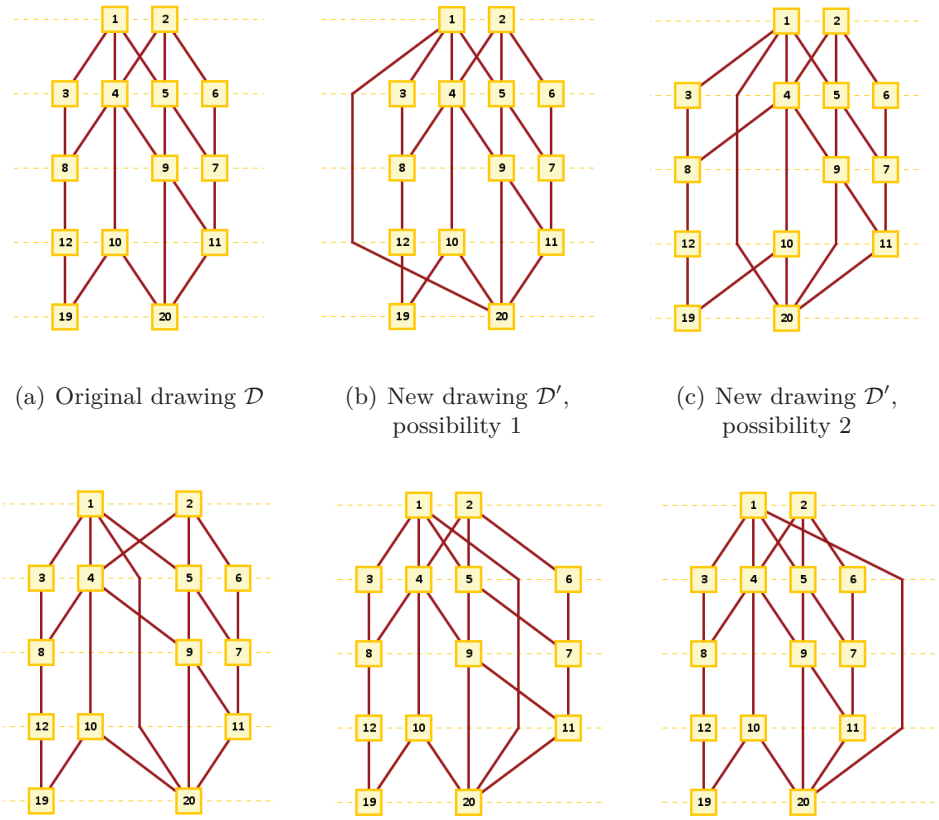
(a) Original drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 5.1: An original drawing (a) and new drawing (b) for inserting a short edge (9,20).



(a) Original drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 5.2: An original drawing (a) and a new drawing (b) for deleting the short edge (17,18).



(a) Original drawing $\mathcal{D}$

(b) New drawing $\mathcal{D}'$, possibility 1

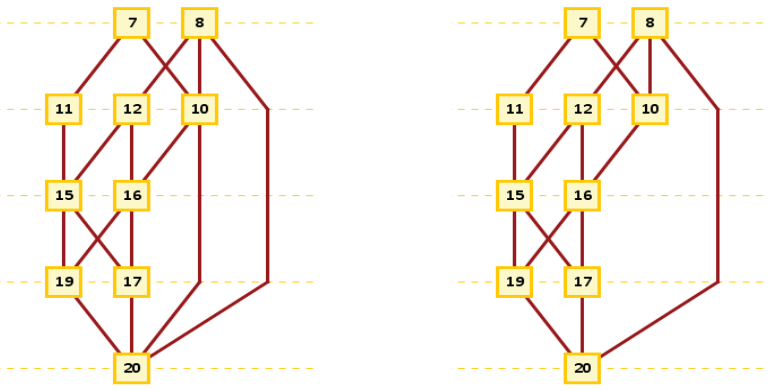
(c) New drawing $\mathcal{D}'$, possibility 2

(d) New drawing $\mathcal{D}'$, possibility 3

(e) New drawing $\mathcal{D}'$, possibility 4

(f) New drawing $\mathcal{D}'$, possibility 5

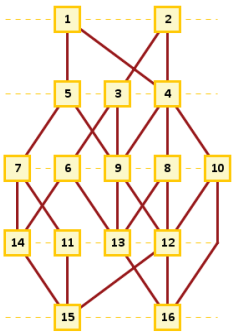
Figure 5.3: An original drawing (a) and 5 new different possible drawings (b)-(f) for inserting a long edge (1,20).



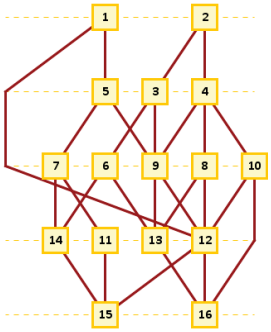
(a) Original drawing $\mathcal{D}$

(b) New drawing $\mathcal{D}'$

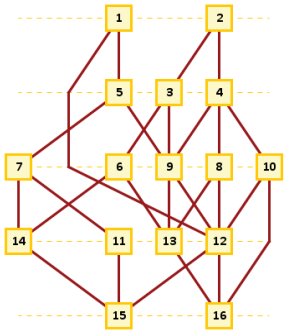
Figure 5.4: Two drawing (a) and new drawing (b) for deleting the long edge (10,20).



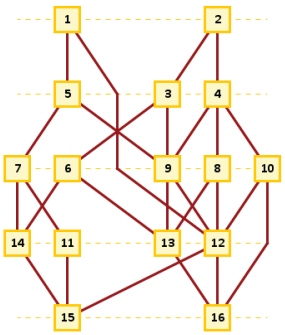
(a) Original drawing $\mathcal{D}$



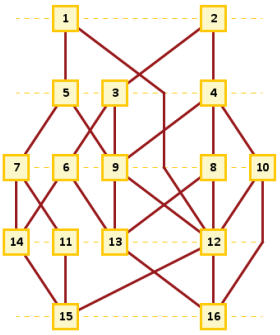
(b) New drawing $\mathcal{D}'$, possibility 1



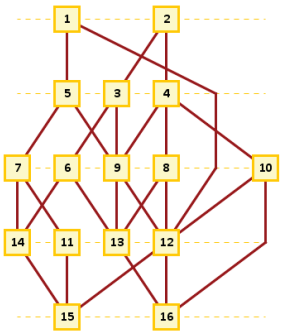
(c) New drawing $\mathcal{D}'$, possibility 2



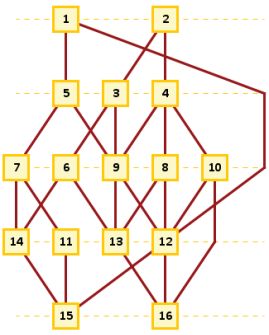
(d) New drawing $\mathcal{D}'$, possibility 3



(e) New drawing $\mathcal{D}'$, possibility 4

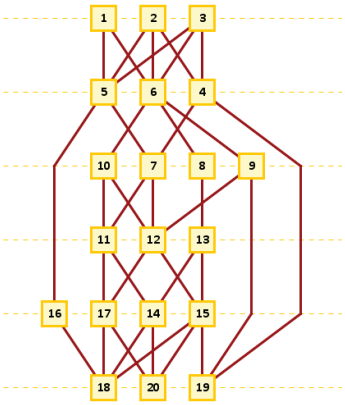


(f) New drawing $\mathcal{D}'$, possibility 5

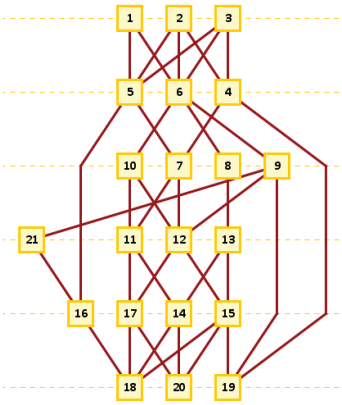


(g) New drawing $\mathcal{D}'$, possibility 6

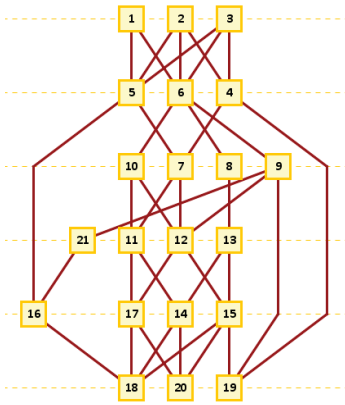
Figure 5.5: An original drawing before modifying edge (a) and 6 new different possible drawings (b)-(g) for modifying the edge (1,4) to (1,12).



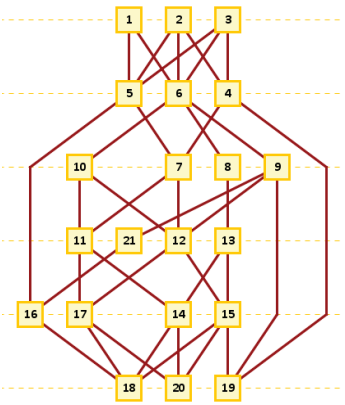
(a) Original drawing $\mathcal{D}$



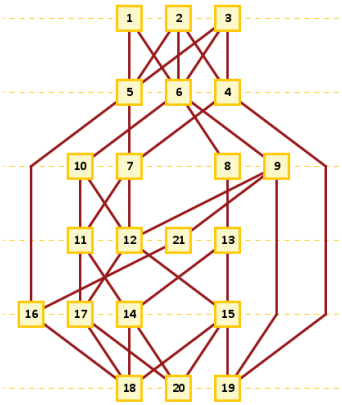
(b) New drawing $\mathcal{D}'$, possibility 1



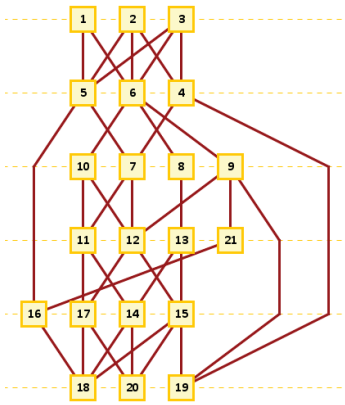
(c) New drawing $\mathcal{D}'$, possibility 2



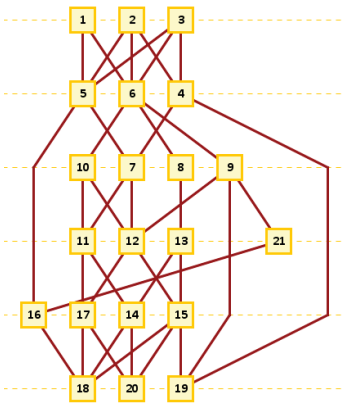
(d) New drawing $\mathcal{D}'$, possibility 3



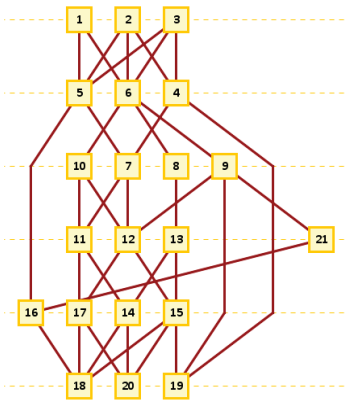
(e) New drawing $\mathcal{D}'$, possibility 4



(f) New drawing $\mathcal{D}'$, possibility 5

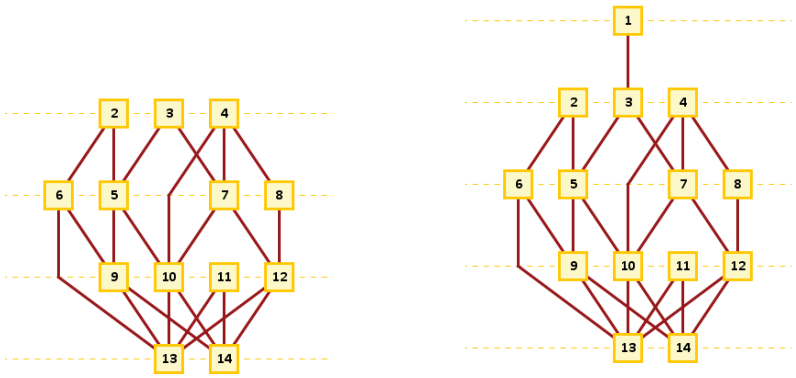


(g) New drawing $\mathcal{D}'$, possibility 6



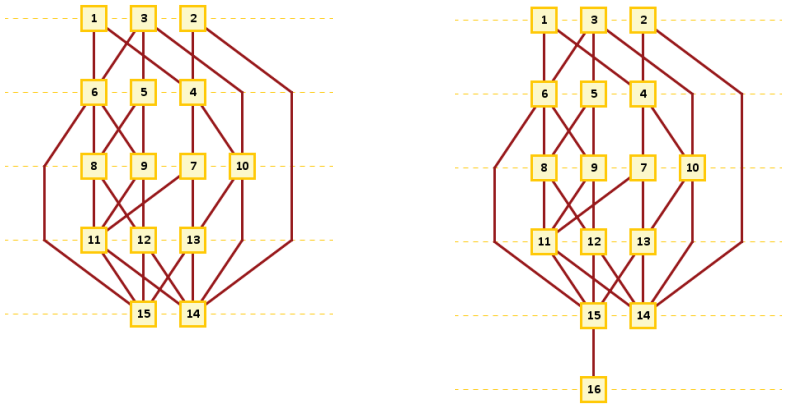
(h) New drawing $\mathcal{D}'$, possibility 7

Figure 5.6: An original drawing (a) and 7 new different possible drawings (b)-(h) for inserting vertex 21 into an existing layer.



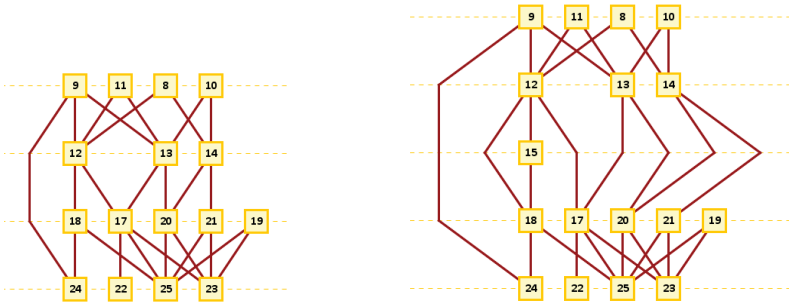
(a) Original drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 5.7: An original drawing (a) and a new drawing (b) for inserting vertex 1 in a new first layer.



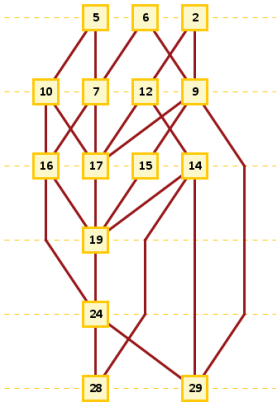
(a) Original drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 5.8: An original drawing (a) and a new drawing (b) for inserting vertex 16 in a new last layer.

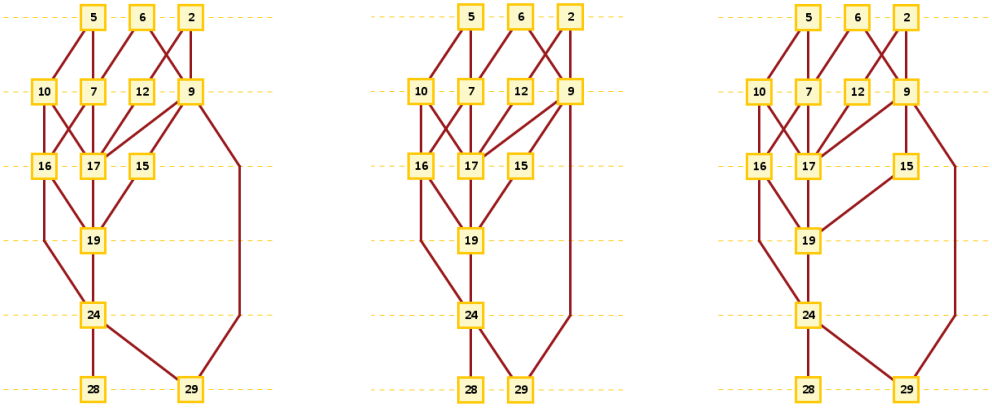


(a) Original drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$

Figure 5.9: An original drawing (a) and a new drawing (b) for inserting vertex 15 in a new intermediate layer.

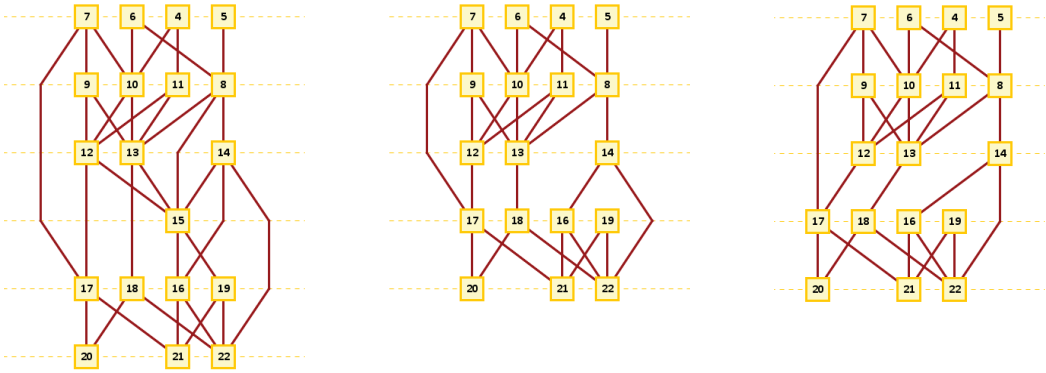


(a) Original drawing $\mathcal{D}$



(b) New drawing $\mathcal{D}'$, possibility 1 (c) New drawing $\mathcal{D}'$, possibility 2 (d) New drawing $\mathcal{D}'$, possibility 3

Figure 5.10: An original drawing (a) and 3 new different possible drawings (b)-(d) for deleting vertex 14 from its layer.

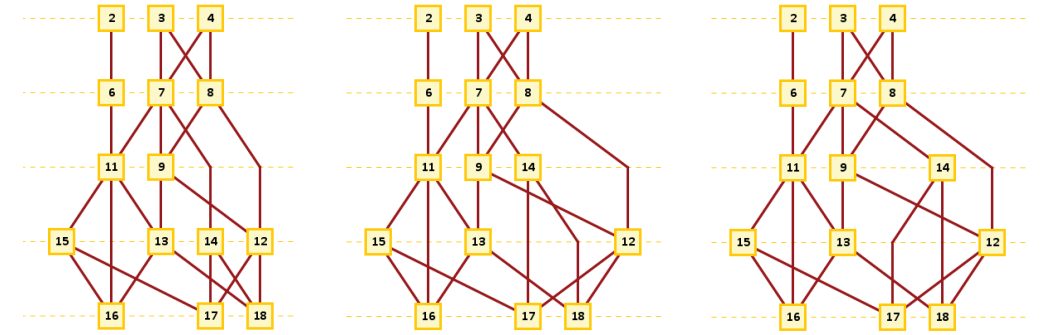


(a) Original drawing $\mathcal{D}$

(b) New drawing $\mathcal{D}'$, possibility 1

(c) New drawing $\mathcal{D}'$, possibility 2

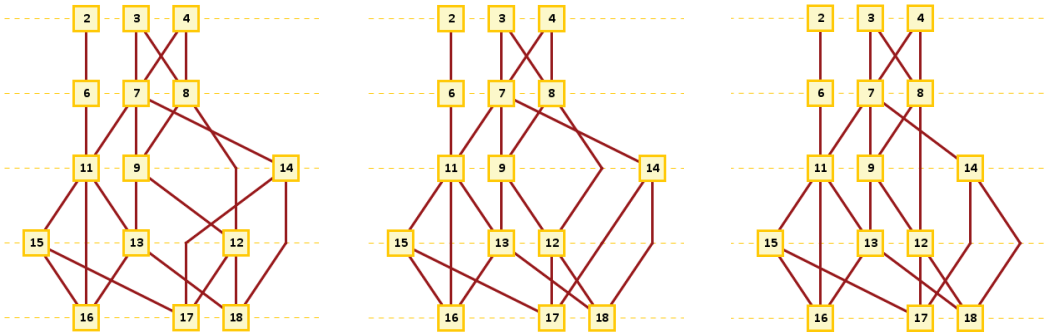
Figure 5.11: An original drawing (a) and 2 new different possible drawings (b) and (c) for deleting vertex 15 and its layer.



(a) Original drawing $\mathcal{D}$

(b) New drawing $\mathcal{D}'$, possibility 1

(c) New drawing $\mathcal{D}'$, possibility 2

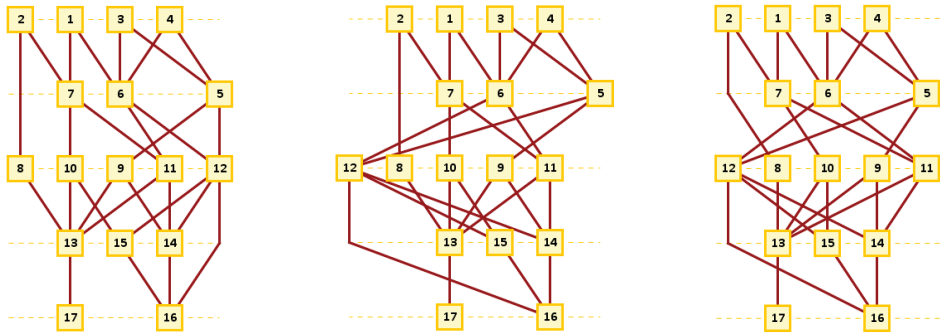


(d) New drawing $\mathcal{D}'$, possibility 3

(e) New drawing $\mathcal{D}'$, possibility 4

(f) New drawing $\mathcal{D}'$, possibility 5

Figure 5.12: An original drawing (a) and 5 new different possible drawings (b)-(f) for modifying vertex 14 layer moving it one layer up.



(a) Original drawing $\mathcal{D}$ (b) New drawing $\mathcal{D}'$, possibility 1 (c) New drawing $\mathcal{D}'$, possibility 2

Figure 5.13: An original drawing (a) and 2 new different possible drawings (b)-(c) for modifying vertex 14 horizontal position in its layer.

5.3 Results

5.3.1 Task 1 Results

We compare the metrics values to the user evaluation values of the similarity between the original drawing $\mathcal{D}$ presented in subfigures (a) in all the 13 figures, and the new drawing(s) $\mathcal{D}'$ presented in the 13 figures starting from subfigure (b). For the 13 graphs, the results for the user and difference metrics for computing the degree of similarity are presented in Figures 5.14-5.26. Note that for the graphs with more than one possible drawing for the executed action, the values of the results are presented in subfigure chart (a) in Figures 5.17, 5.18-5.19, and 5.23-5.26. The 20 participant's degrees of similarity for the 13 graphs are given in the two Tables 5.2 and 5.3.

We consider a metric value to be near to the user evaluation if the metric value is $\pm 10\%$ of the user evaluation, otherwise, the metric value is not considered. The percentage values of the considered metrics to be near of the user evaluations are presented in Table 5.1.

5.3.2 Task 2 Results

For the graphs with more than one possible drawing for the executed action (Figures 5.17, 5.18-5.19, and 5.23-5.26), the user ordering for these possibilities are given in the subfigure chart (b) in the figures. We have originally ordered these possible drawings by inserting the new vertices and edges outside the drawing (starting from the left border of the drawing) and then go smoothly inside the drawing until they reach the right-hand border. In each of these (b) subfigures, the presented order of the possibilities is presented with the percentage of the users who have given that rank.

Table 5.1: Percentage values of the average metrics values and their relation to the user evaluation of the degree of similarity ±10% for the 36 new drawings to their original drawings given in Figures 5.1-5.13.

Difference Metric	Percentage
Edge length $\mu_{\mathcal{EL}}$	86%
Vertex minimum angle $\mu_{\mathcal{V}\vartheta}$	64%
Edge orthogonality $\mu_{\mathcal{E}\delta}$	58%
Euclidean distance $\mu_{\mathcal{ED}}$	19%
Relative distance $\mu_{\mathcal{RD}}$	14%
Edge crossing $\mu_{\mathcal{EC}}$	4%
No metrics near user evaluation	3%

For example, in Figure 5.17, drawing 5 has been selected (by 25% of the participants) as the most similar drawing to the original one, drawing 3 has been selected (by 35% of the participants) as the second most similar drawing to the original one, drawing 4 has been selected (by 25% of the participants) as the third most similar drawing to the original one, drawing 2 has been selected (by 40% of the participants) as the fifth most similar drawing to the original one, and drawing 1 has been selected (by 25% of the participants) as the most different drawing to the original one. The different orderings by the 20 participants for the Figures 5.17, 5.18-5.19, and 5.23-5.26 are given in Table 5.4.

We compute the relative ordering produced by each difference metric and compare this ordering with the user ordering. The number of shifts required to transform an ordering of a metric into a user ordering is computed for each graph. The results for these numbers of shifts for each metric are given in the subfigure (c) in Figures 5.17, 5.18-5.19, and 5.23-5.26.

Furthermore, we found that 72% of the users follow the order of inserting new vertices and edges starting from the right border of the drawing and then go gradually inside the drawing till reaching the left boundary. Also, we found that inserting the new vertices and edges away from the center of the drawing leads to lower change in most of the difference metrics, which is reflects that the change between both drawings is smaller when more vertices and edges keep the same geometric values.

5.4 Summary

In this chapter we introduced the results of an experimental user study. This aim of this user study is to validate the difference metrics introduced in Section 4.3 depending on user evaluation of the similarity between two drawings of a dynamic hierarchical graph. From this user study we found that, in 97% of the examples, the values of the difference metrics are in the range of ± 10% of the user evaluation of the similarity measure. In case of more than one possible drawing for executing an action, inserting the new vertices and edges away from the center of the drawing leads to lower change in most of the difference metrics. This behaviour completely reflects that the change between both drawings is smaller when more vertices and edges keep the same geometric values.

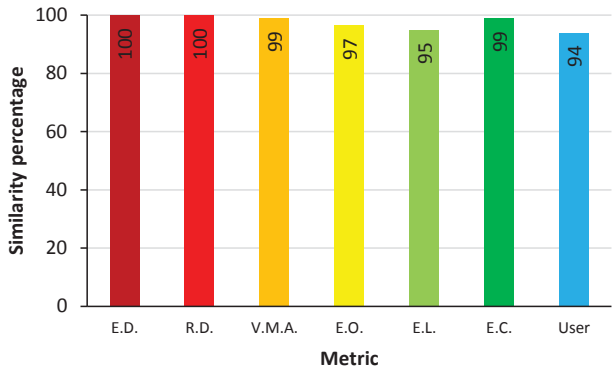


Figure 5.14: Difference metrics and user degree of similarity values for the two drawings presented in Figure 5.1, where the action is to insert a short edge (9,20).

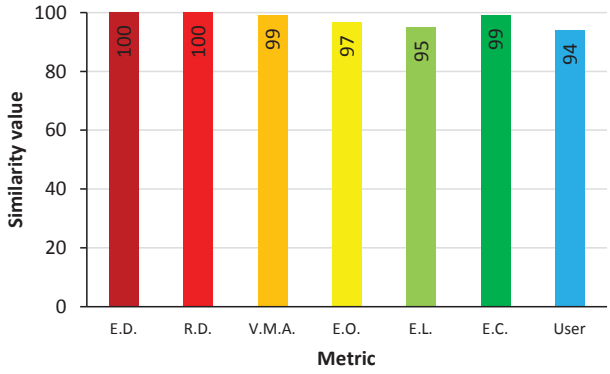


Figure 5.15: Difference metrics and user degree of similarity values for the two drawings presented in Figure 5.2, where the action is to delete the short edge (17,18).

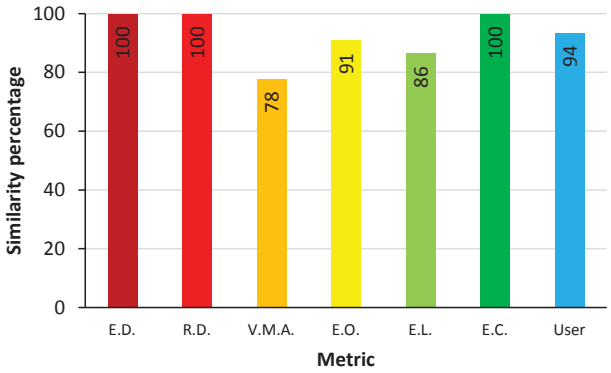
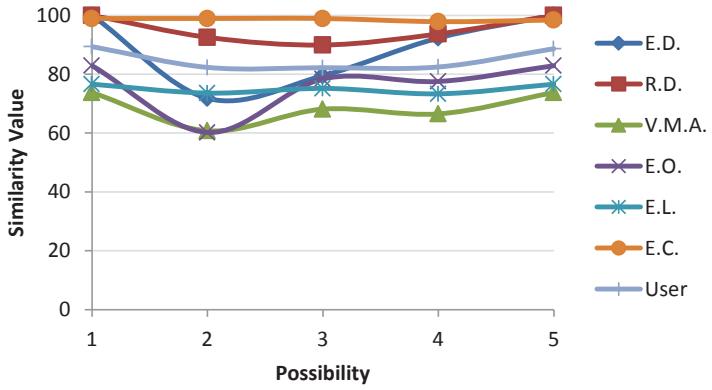
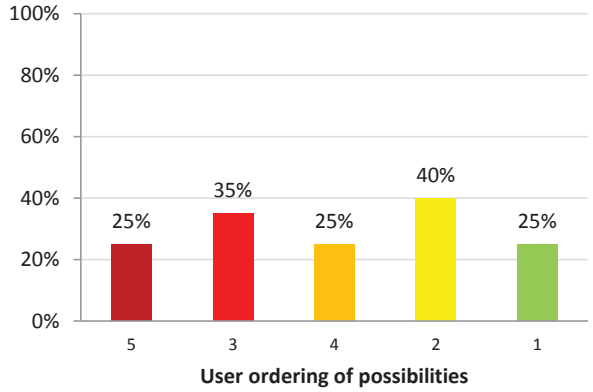


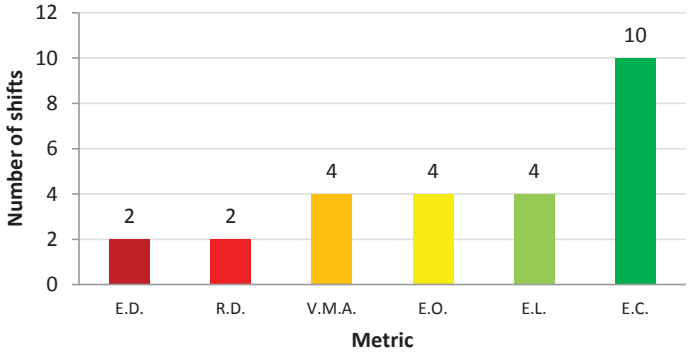
Figure 5.16: Difference metrics and user degree of similarity values for the two drawings presented in Figure 5.4, where the action is to delete the long edge (10,20).



(a)

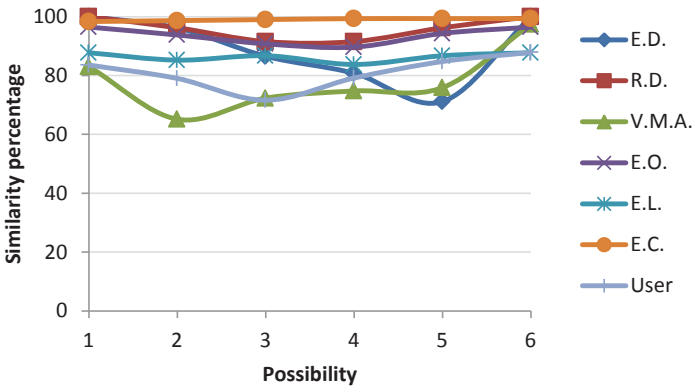


(b)

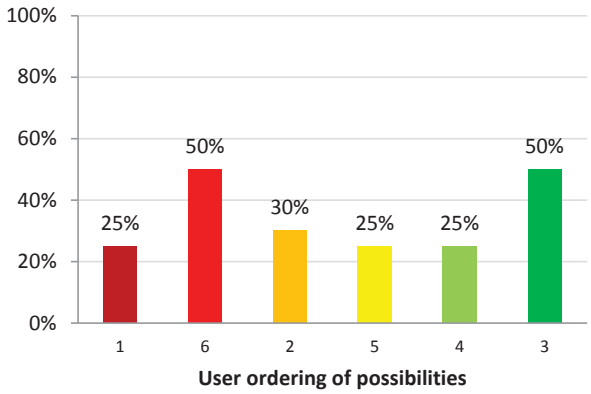


(c)

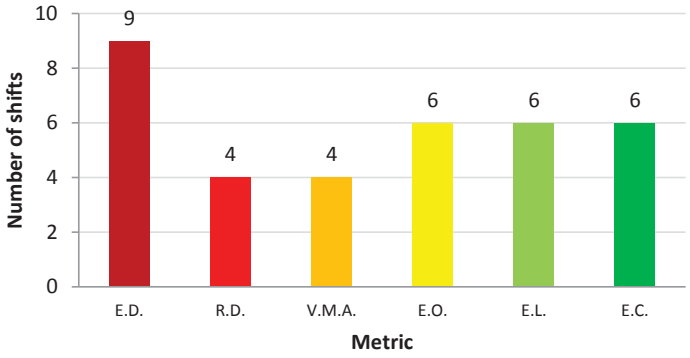
Figure 5.17: Results for Figure 5.3 for inserting a long edge (1,20). (a) Averaged values of difference metrics and user evaluations of the degree of similarity between each of the 5 new possible drawings presented in Figures 5.3(b)-5.3(f) to the original one in Figure 5.3(a). (b) Percentage values of each possibility of its order according to the user ordering. (c) Number of ordering shifts each metric needs to have the user ordering.



(a)

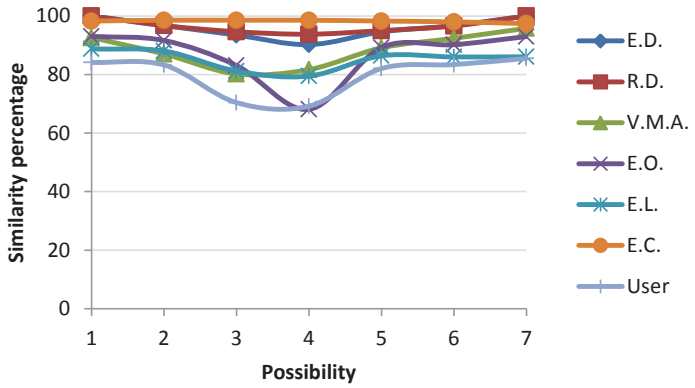


(b)

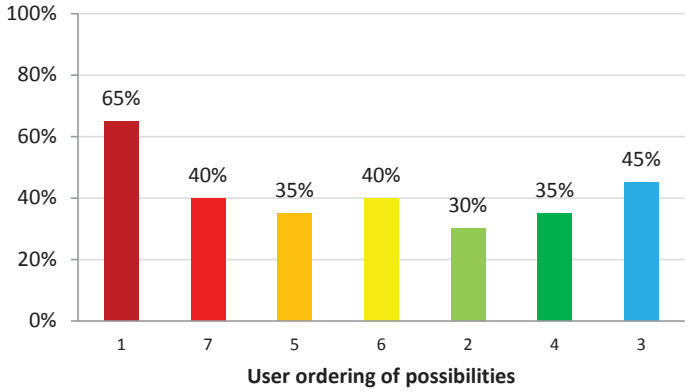


(c)

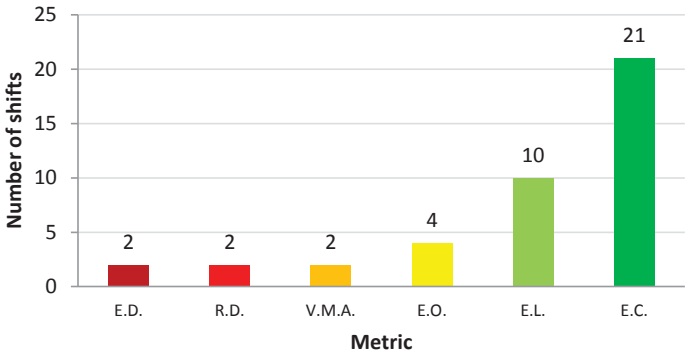
Figure 5.18: Results for Figure 5.5 for modifying the edge (1,4) to (1,12). (a) Averaged values of difference metrics and user evaluations of the degree of similarity between each of the 6 new possible drawings presented in Figures 5.5(b)-5.5(g) to the original one given in Figure 5.5(a). (b) Percentage values of each possibility of its order. (c) Number of ordering shifts each metric needs to have the user ordering.



(a)



(b)



(c)

Figure 5.19: Results for Figure 5.6 for inserting vertex 21 into an existing layer. (a) Averaged values of difference metrics and user evaluations of the degree of similarity between each of the 7 new possible drawings presented in Figures 5.6(b)-5.6(h) to the original one given in Figure 5.6(a). (b) Percentage values of each possibility of its order. (c) Number of ordering shifts each metric needs to have the user ordering.

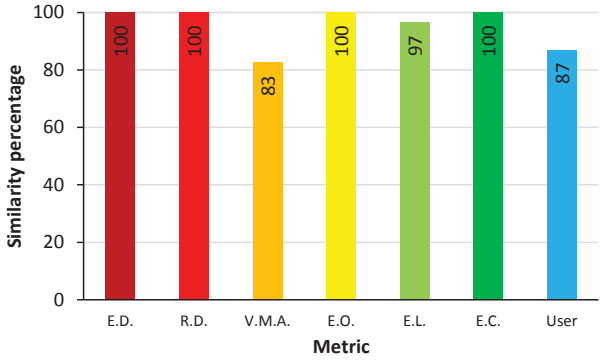


Figure 5.20: Difference metrics and user degree of similarity values for the two drawings presented in Figure 5.7, where the action is to insert the new vertex 1 into the new first layer.

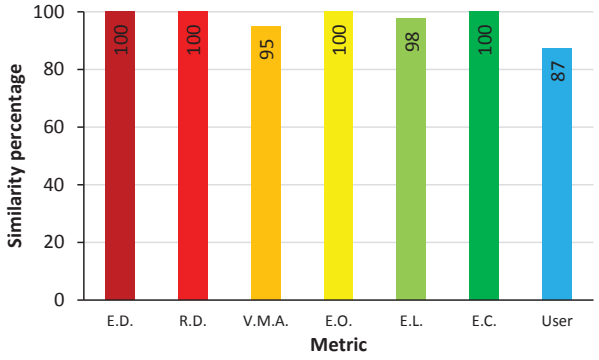


Figure 5.21: Difference metrics and user degree of similarity values for the two drawings presented in Figure 5.8, where the action is to insert the new vertex 16 into the new last layer.

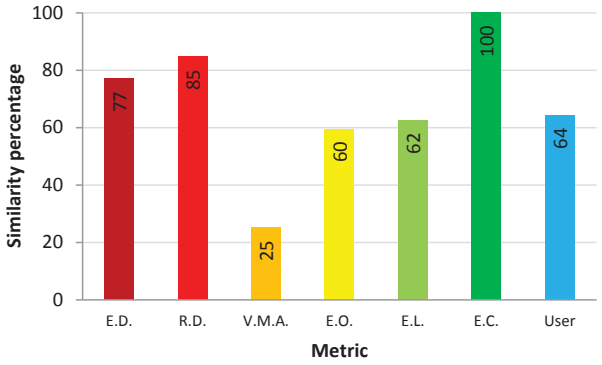
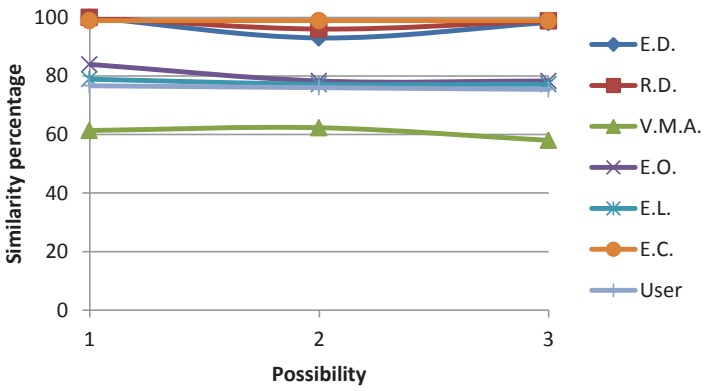
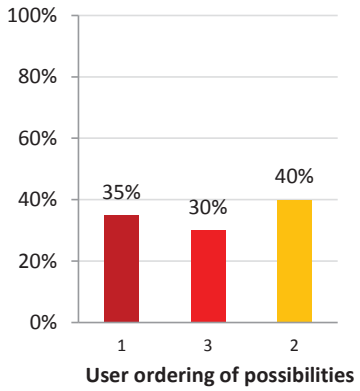


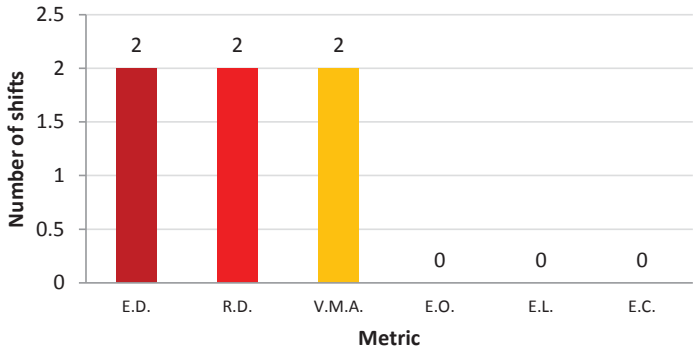
Figure 5.22: Difference metrics and user degree of similarity values for the two drawings presented in Figure 5.9, where the action is to insert the new vertex 15 into the new intermediate layer.



(a)

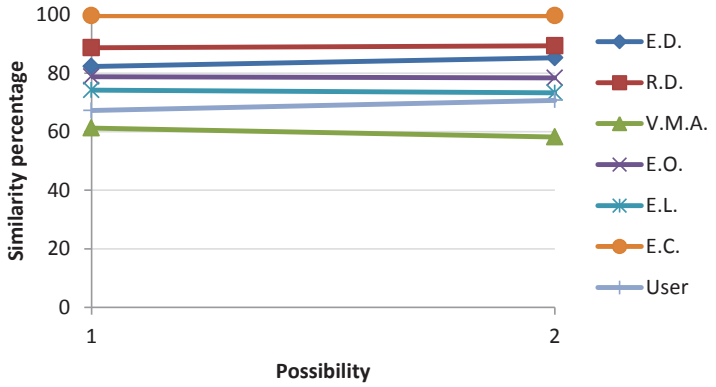


(b)

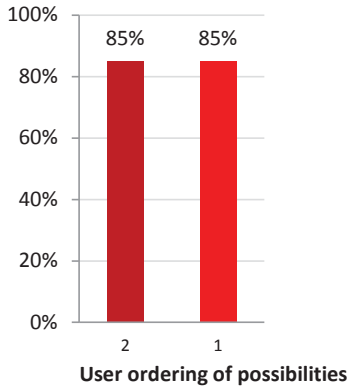


(c)

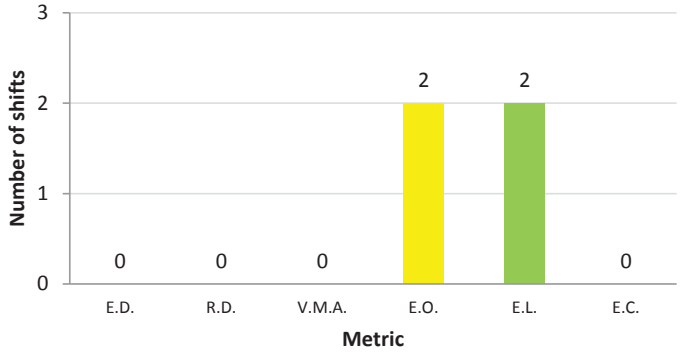
Figure 5.23: Results for Figure 5.10 for deleting the vertex 14 from its layer. (a) Averaged values of difference metrics and user evaluations of the degree of similarity between each of the 7 new possible drawings presented in Figures 5.10(b)-5.10(d) to the original one given in Figure 5.10(a). (b) Percentage values of each possibility of its order. (c) Number of ordering shifts each metric needs to have the user ordering.



(a)

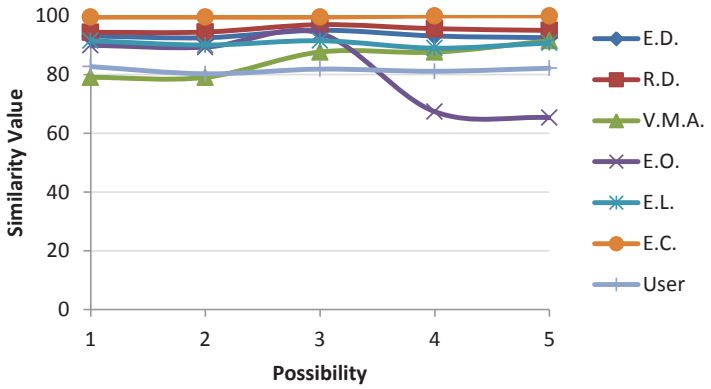


(b)

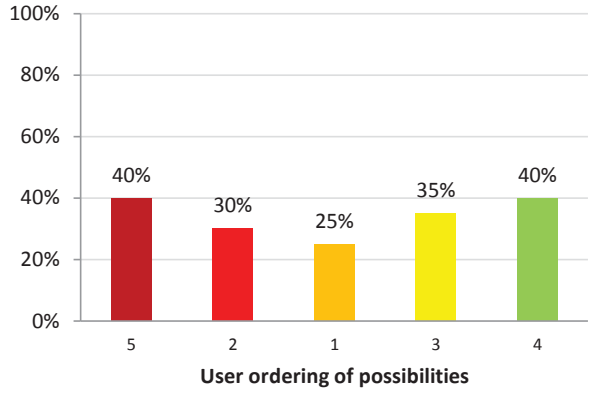


(c)

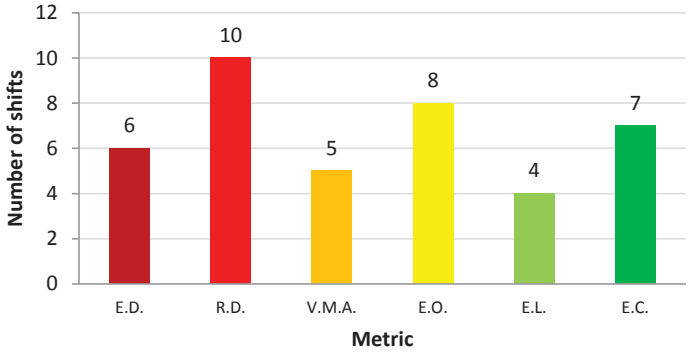
Figure 5.24: Results for Figure 5.11 for deleting the vertex 21 and its layer. (a) Averaged values of difference metrics and user evaluations of the degree of similarity between each of the 2 new possible drawings presented in Figures 5.11(b)-5.11(c) to the original one given in Figure 5.11(a). (b) Percentage values of each possibility of its order. (c) Number of ordering shifts each metric needs to have the user ordering.



(a)

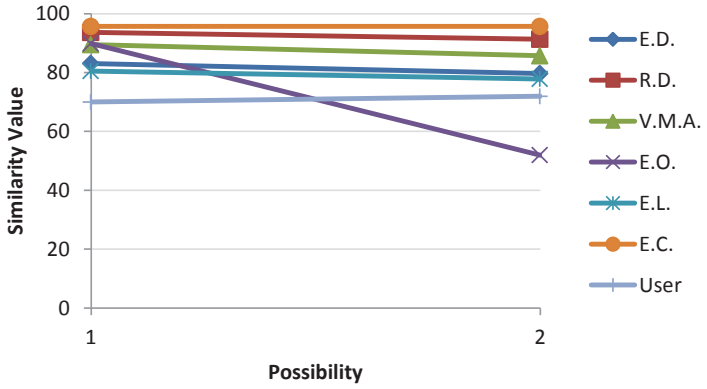


(b)

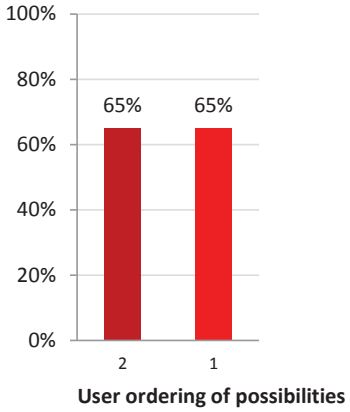


(c)

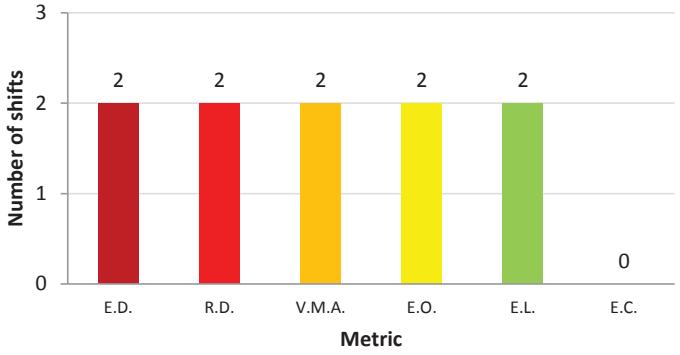
Figure 5.25: Results for Figure 5.12 for modifying vertex 14 layer by moving it one layer up. (a) Averaged values of difference metrics and user evaluations of the degree of similarity between each of the 5 new possible drawings presented in Figures 5.12(b)-5.12(f) to the original one given in Figure 5.12(a). (b) Percentage values of each possibility of its order. (c) Number of ordering shifts each metric needs to have the user ordering.



(a)



(b)



(c)

Figure 5.26: Results for Figure 5.13 for modifying vertex 14 horizontal position in its layer. (a) Averaged values of difference metrics and user evaluations of the degree of similarity between each of the 2 new possible drawings presented in Figures 5.13(b)-5.13(c) to the original one given in Figure 5.13(a). (b) Percentage values of each possibility of its order. (c) Number of ordering shifts each metric needs to have the user ordering.

Table 5.2: User percentage values for the degree of similarity of the new and original drawing(s) in Figures 5.1 - 5.6.

user id	Figure 5.1	Figure 5.2	Figure 5.3					Figure 5.4	Figure 5.5						Figure 5.6						
			1	2	3	4	5		1	2	3	4	5	6	1	2	3	4	5	6	7
1	95	95	95	95	95	95	95	95	90	90	90	90	95	95	70	90	90	90	90	90	90
2	92	82	95	82	78	81	93	98	80	88	64	83	74	90	75	75	65	60	82	85	94
3	75	100	90	65	95	80	95	90	80	70	70	95	90	95	90	95	70	70	90	80	85
4	95	95	95	90	95	90	95	95	90	90	85	85	90	95	90	90	70	75	95	93	93
5	96	100	98	97	95	96	98	96	96	98	90	95	95	96	95	90	85	90	92	97	93
6	90	90	80	90	70	75	90	90	60	65	40	50	70	80	65	68	60	60	60	60	70
7	80	90	90	85	90	80	85	80	70	65	65	65	70	75	83	82	70	69	79	85	88
8	85	100	85	60	60	75	80	90	65	55	40	65	80	80	60	65	55	25	65	75	80
9	90	80	80	70	70	70	80	90	70	60	70	60	70	70	80	70	60	60	70	80	90
10	95	96	92	91	90	91	93	94	90	86	85	90	91	91	85	90	88	90	93	94	90
11	98	98	80	85	85	87	88	90	85	92	85	82	85	80	85	85	70	95	90	90	90
12	92	95	95	50	45	60	95	95	90	85	40	45	80	95	90	90	80	35	70	85	80
13	99	99	99	99	99	99	99	100	98	98	98	98	98	98	95	95	95	95	95	95	95
14	95	100	90	80	80	80	90	100	90	60	50	60	70	95	80	70	50	60	90	90	80
15	95	100	85	85	80	80	80	95	90	90	90	90	90	90	85	85	60	90	90	85	90
16	85	100	75	65	65	70	85	90	75	55	50	75	80	85	80	65	55	65	80	75	60
17	95	95	95	95	95	95	95	95	90	90	90	95	95	90	90	90	60	70	80	80	98
18	95	80	85	70	70	60	70	95	80	70	50	80	90	85	95	90	40	30	60	70	80
19	85	85	85	95	90	90	70	95	85	85	85	85	85	75	85	95	95	95	75	65	65
20	98	98	98	97	97	95	97	98	97	88	95	95	97	97	95	95	90	90	95	95	95
Average	**92**	**94**	**89**	**82**	**82**	**83**	**89**	**94**	**84**	**79**	**72**	**79**	**85**	**88**	**84**	**83**	**70**	**69**	**82**	**83**	**86**
Min.	75	80	75	65	45	70	70	80	60	55	40	45	70	70	60	65	55	25	60	60	60
Max.	99	100	99	99	99	99	99	99	100	98	98	95	95	98	98	95	95	95	95	95	95

Table 5.3: User percentage values for the degree of similarity of the new and original drawing(s) in Figures 5.7 - 5.13.

user id	Figure 5.7	Figure 5.8	Figure 5.9	Figure 5.10 1	Figure 5.10 2	Figure 5.10 3	Figure 5.11 1	Figure 5.11 2	Figure 5.12 1	Figure 5.12 2	Figure 5.12 3	Figure 5.12 4	Figure 5.12 5	Figure 5.13 1	Figure 5.13 2
1	90	90	70	70	70	85	70	75	90	93	90	87	90	95	95
2	93	93	70	93	93	65	75	70	89	80	74	78	87	69	74
3	95	95	60	80	80	80	60	80	90	85	85	90	95	100	80
4	93	93	70	85	80	85	85	90	85	80	90	90	90	80	80
5	93	95	80	92	90	90	92	96	95	95	95	100	100	95	100
6	70	70	40	65	68	60	40	60	100	100	100	100	100	50	80
7	90	85	80	80	75	73	60	62	65	70	70	70	80	75	70
8	95	95	55	40	50	65	40	60	80	75	80	75	70	50	40
9	90	90	70	70	60	80	60	70	60	60	60	60	60	50	50
10	90	92	85	93	94	90	89	91	90	91	89	90	92	91	90
11	75	80	40	80	80	85	85	85	96	92	85	85	90	60	50
12	90	90	30	85	75	60	60	60	70	60	80	80	75	70	70
13	95	95	50	95	95	95	60	60	95	95	95	95	95	100	100
14	70	70	50	90	90	70	60	60	90	80	85	85	85	90	95
15	90	90	90	85	90	90	99	80	75	75	75	75	75	70	70
16	70	75	40	45	40	55	45	30	65	60	75	80	70	55	85
17	90	90	70	70	70	70	70	80	80	80	80	70	60	40	50
18	95	95	85	85	90	85	50	60	85	80	80	70	90	40	40
19	70	70	60	40	40	40	55	55	60	65	60	50	55	40	40
20	95	95	90	90	90	85	90	90	95	90	90	93	85	80	80
Average	**87**	**87**	**64**	**77**	**76**	**75**	**67**	**71**	**83**	**80**	**82**	**81**	**85**	**70**	**72**
Min.	70	70	40	40	40	40	40	55	60	60	60	50	55	40	40
Max.	95	95	90	95	95	95	99	96	100	100	100	100	100	100	100

Table 5.4: User ordering of the different possible drawings presented in Figures 5.3, 5.5, 5.6, 5.10, 5.11, 5.12, and 5.13.

user id	Figure 5.3	Figure 5.5	Figure 5.6	Figure 5.10	Figure 5.11	Figure 5.12	Figure 5.13
1	3, 4, 5, 2, 1	4, 6, 2, 5, 1, 3	2, 6, 5, 1, 7, 3, 4	1, 3, 2	2, 1	2, 5, 3, 1, 4	2, 1
2	3, 2, 1, 5, 4	1, 6, 4, 2, 3, 5	5, 7, 1, 6, 2, 4, 3	3, 2, 1	1, 2	5, 3, 1, 4, 2	2, 1
3	3, 4, 5, 2, 1	2, 5, 1, 6, 3, 4	1, 7, 6, 5, 2, 4, 3	2, 1, 3	2, 1,	5, 4, 1, 3, 2	1, 2
4	5, 3, 4, 1, 2	6, 4, 5, 1, 2, 1	1, 2, 5, 6, 7, 3, 4	2, 1, 3	2, 1	5, 3, 2, 1, 4	1, 2
5	2, 4, 3, 5, 1	5, 6, 3, 1, 2, 4	1, 2, 5, 6, 3, 4, 7	1, 3, 2	2, 1	2, 1, 3, 5, 4	2, 1
6	5, 3, 4, 1, 2	5, 6, 4, 2, 1, 3	7, 6, 1, 2, 4, 5, 3	3, 1, 2	2, 1	3, 4, 5, 1, 2	2, 1
7	5, 1, 2, 4, 3	6, 1, 2, 4, 5, 3	1, 2, 5, 6, 7, 4, 3	1, 3, 2	2, 1	2, 1, 5, 4, 3	2, 1
8	4, 1, 5, 2, 3	5, 4, 6, 2, 1, 3	6, 7, 1, 2, 5, 3, 4	3, 1, 2	2, 1	2, 5, 1, 4, 3	2, 1
9	1, 5, 3, 2, 4	1, 6, 2, 5, 3, 4	1, 7, 5, 6, 2, 3, 4	1, 2, 2	1, 2	1, 2, 5, 3, 4	1, 2
10	5, 1, 2, 3, 4	1, 6, 5, 2, 4, 3	1, 2, 7, 5, 6, 3, 4	3, 2, 1	1, 2	5, 1, 2, 3, 4	2, 1
11	4, 3, 5, 2, 1	4, 5, 2, 6, 1, 3	1, 7, 6, 5, 2, 4, 3	1, 3, 2	2, 1	5, 3, 1, 2, 4	2, 1
12	5, 1, 4, 2, 3	5, 4, 6, 1, 2, 3	5, 3, 2, 4, 1, 7, 6	3, 1, 2	2, 1	3, 4, 5, 1, 2	1, 2
13	1, 5, 4, 2, 3	1, 6, 2, 5, 4, 3	1, 7, 6, 5, 2, 4, 3	1, 2, 3,	4, 1	1, 5, 4, 2, 3	1, 2
14	2, 3, 1, 4, 5	1, 6, 5, 2, 4, 3	1, 7, 5, 6, 2, 4, 3	3, 4, 2	2, 1	5, 1, 2, 4, 3	2, 1
15	3, 4, 1, 2, 5	2, 3, 4, 5, 6, 1	6, 1, 7, 5, 4, 2, 3	3, 1, 2	2, 1	2, 1, 4, 3, 5	2, 1
16	2, 3, 4, 5, 1	6, 4, 1, 5, 3, 2	1, 3, 7, 2, 6, 5, 4	2, 3, 1	2, 1	5, 1, 4, 2, 3	2, 1
17	4, 2, 1, 5, 3	4, 6, 5, 1, 2, 3	1, 7, 2, 5, 6, 4, 3	2, 3, 1	2, 1	5, 2, 3, 1, 4	2 ,1
18	1, 3, 5, 4, 2	5, 6, 1, 4, 2, 3	1, 2, 3, 4, 6, 5, 7	2, 3, 1	2, 1	1, 5, 2, 3, 4	1, 2
19	2, 4, 3, 1, 5	3, 4, 5, 2, 6, 1	3, 4, 5, 6, 2, 1, 7	2, 1, 3	2, 1	1, 2, 4, 3, 5	1, 2
20	1, 3, 2, 5, 4	6, 5, 1, 4, 3, 2	1, 2, 7, 6, 5, 3, 4	1, 2, 3	2, 1	3, 5, 4, 2, 1	2, 1

6 Conclusion and Outlook

Der gerade Weg ist der kürzeste, aber es dauert meist am längsten, bis man auf ihm zum Ziele gelangt.[1]

This thesis deals with dynamic hierarchical graph drawing. The core contribution of the thesis consists in proposing and validating difference metrics to measure the difference between two drawings of a dynamically changing hierarchical graph. Moreover, we study the problem of edge crossing minimization in hierarchical graphs. A final concerned point is on generating random hierarchical graphs controlling all the topological properties of hierarchical graphs.

6.1 Conclusion

In the following, the main contributions of this thesis are explicitly formulated.

6.1.1 Minimize Crossings in Hierarchical Graph Drawing

The number of crossings in a hierarchical graph drawing depends only on the order of the vertices in their layer (as a topological entity) not on their absolute positions. After an intensive comparison between the one-sided crossing minimization techniques, we introduced the efficient barycenter algorithm in Section 3.4. This algorithm computes the x-coordinates

[1] Georg Christoph Lichtenberg. English translation: The straight path is the shortest, but it usually takes the longest time, to reach your target.

of the vertices in each layer in an efficient way. Firstly, it computes the barycenter value for every vertex in the i^{th} layer as the average of the x-coordinates of its neighbours in the two $(i-1)^{th}$ and $(i+1)^{th}$ layers. Then, before moving to the next layer $(i+1)$ (the $(i+1)^{th}$ layer) to compute their barycenter value as normal, it sorts the vertices in layer i according to their recent barycenter values and then directly set their x-coordinates as their ranks after the sorting step. Computing the x-coordinates in this way leads to a lower number of crossings in addition to faster convergence. The experiments yield an improvement of around 4% in the number of crossings over the recent global sifting algorithm and around 15% over the other layer-by-layer one-sided 2-layer heuristics, specially for small and sparse graphs.

6.1.2 Generating random hierarchical graphs

Since any hierarchical graph is originally a directed acyclic graph, the upper bound of number of edges in a hierarchical graph depends on the layering of the vertices. This means that we could not get a precise upper bound of the number of edges before assigning the vertices to the layers. This completely affects the edge density of the considered graph. The usually considered way to deal with random hierarchical graphs is to generate a random directed acyclic graph and then transform it into a hierarchical graph through the Sugiyama approach. In contrast to this, in our approach a layered hierarchical graph is generated directly. We could derive a formula of the maximum number of edges in a hierarchical graph and then we could control precisely the edge density, see Section 3.7. We introduced two generators in order to randomly generate either proper or general hierarchical graphs. These two generators control all the parameters of hierarchical graphs like, the number of layers, the number of vertices in each layer, the number of edges going out of a vertex, the edge density, the long edges ratio to the short ones (in general non-proper hierarchical graphs).

6.1.3 Difference metrics for dynamic hierarchical graph drawing

In the perspective of drawing dynamic hierarchical graphs we have been concerned with the following points:

- We study the different cases of executing an action on a hierarchical graph in a dynamic scenario. The possible actions could be extracted from the product:

$$\{insert,\ delete,\ modify\} \times \{vertex,\ edge\}$$

  When an action $\mathcal{A}$ is applied to a hierarchical graph $\mathcal{H}$, a new graph $\mathcal{H}'$ is produced. Based on the topological and structural properties of hierarchical graphs, the action product induces 13 possible executed actions, as presented in Section 4.3.1.

- Furthermore, we introduced a new general form for computing a quantitative measure of the difference between two drawings of a dynamic hierarchical graph when executing an action. The proposed form depends on distinguishing between the graph components in both current drawing $\mathcal{D}(\mathcal{H})$ and new drawing $\mathcal{D}'(\mathcal{H}')$. The vertices and edges that will be inserted to the new drawing $\mathcal{D}'(\mathcal{H}')$ or deleted from the current $\mathcal{D}(\mathcal{H})$ are called *action components*, where the remaining vertices and edges that are found in both drawings $\mathcal{D}(\mathcal{H})$ and $\mathcal{D}'(\mathcal{H}')$ are called *shared components*. Since each shared component has a

geometric representation in each of the two drawings $\mathcal{D}$ and $\mathcal{D}'$, the difference metric value of the shared components is computed as the sum of the difference metric values of each shared component in $\mathcal{D}$ and $\mathcal{D}'$. Moreover, as each action component has just one representation in either $\mathcal{D}$ or $\mathcal{D}'$, the difference metric value of the action components is computed as the difference between the two sums of the action component's metric values in each drawing $\mathcal{D}$ and $\mathcal{D}'$. The total difference metric value is sum of the shared and action component's metric values. Finally, the difference metric value is scaled against a maximum value of the considered metric in drawing $\mathcal{D}$ (based on the definition of the mental map) in order to get a normalized value of the metric between 0 and 1. Scaling the metrics in this way ensures that the metric value does not depend on the nature of the underlying graph [Pur02].

- Some difference metrics formulas have been used, in [BT00], to measure the difference between two orthogonal drawings of the same graph. Furthermore, formal metrics for measuring the aesthetic presence in a graph drawing for some common aesthetic criteria are presented in [Pur02]. In Section 4.3, we use the proposed general form of difference metrics of a dynamic hierarchical graph to extend these metrics forms presented in [BT00, Pur02].

6.1.4 Crossing minimization in dynamic hierarchical graph drawing

We introduced the problem of minimizing edge crossings when the hierarchical graph is drawn in a dynamic environment. In other words, the crossing minimization problem is $\mathcal{NP}$-hard and solved normally using iterative heuristic techniques. In dynamic scenarios, we have to minimize the difference between the current drawing and the new one after executing an action on a hierarchical graph. The question now is: which drawing ${\mathcal{D}'}_*$ from the s ones of the s iterations produced during the running of the crossing minimization heuristic ${\mathcal{D}'}_0, {\mathcal{D}'}_1, \cdots, {\mathcal{D}'}_s$ that has the minimum number of crossings and in the same time has the minimum change to the current one $\mathcal{D}$? We propose two solutions for this problem. The first solution is to consider that the required ${\mathcal{D}'}_*$ should have some predefined aesthetic values of some aesthetic drawing criteria (as presented in [Pur02]). The second solution is that the difference metric values of the required drawing ${\mathcal{D}'}_*$ and the original drawing $\mathcal{D}$ should not exceed some predefined minimum difference metric values.

6.1.5 Experimental user study on dynamic hierarchical graph drawing

In order to validate the proposed difference metrics forms that measure the change between the current and the new drawing after executing an action, we performed an experimental user study. The user study has been implemented during the annual meeting *Dagstuhl Klausurtagung "LST Schmeck" 2011* in *Schloss Dagstuhl*, 16-18 November 2011. The participants have been asked two questions: the first regards the degree of similarity between two drawings after executing an action, and the second regards ordering some possible drawings of an action according to the similarity to their original drawing. This study proposed a statistical validation of the difference metrics introduced in Section 4.3. The results of the user study have been presented in Section 5.3.

6.2 Outlook

In addition to the work we have done in the area of drawing dynamic hierarchical graphs and minimizing edge crossings of hierarchical graphs, there a space for many extensions and several future points could be investigated. Possible improvements of the presented work in this thesis include the following.

- Although some work has been carried out in combining more than one step of the Sugiyama approach [UBSE98, BBG11, CGMW11] in order to get more aesthetic drawing, as presented in Section 3.6, more investigation in this direction should be considered.
- For the problem of hierarchical graphs crossing minimization, a perspective of using random sequence of the layers could be considered. Also, a random selection of the vertices in the whole graph; nevertheless the layer sequence could be considered although this may increase the running time [SNM99, BKS10].
- Regarding the problem of minimizing edge crossings of dynamic hierarchical graphs (Section 4.4), an extension to the problem could be done by considering the problem as a multi-objective optimization model. Hence, some multi-objective heuristic techniques could be applied [Deb01, DF07].
- More detailed user studies should be implemented with respect to the user's effort of extracting the differences between two drawings of a dynamic hierarchical graph. This could be done by computing the time the user spends in order to recognize the executed action, as some user studies introduced in [Pur00, KG06, SBOP08].
- An interesting problem of drawing dynamic hierarchical graphs is drawing the graph in a restricted-width area. In this case, we may be forced to move some vertex from its layer to some upper or lower layers in order to get a free position for inserting a new vertex in this layer. Unfortunately, modifying shared vertices layers contradicts with the second condition introduced by Branke [Bra01] (presented in Section 4.3). Moving some vertices to a different layer gives the chance to follow the condition of the restricted width of the drawing instead of exceeding it.
- In a *radial drawing* of a hierarchical graph [BFF05, Bac07, DDL07, BBBH11], the vertices are placed on concentric circles rather than on horizontal lines and the edges are drawn as outward monotone segments of spirals. So, it seems natural to study the following insights in radial drawings of hierarchical graphs:
  - Considering the way of executing an action on a hierarchical graph with that drawn radially. Furthermore, the different possibility of the executed actions should be investigated. Drawing some difference metrics based on the properties of the radial drawing.
  - Trying to consider some difference metrics based on the properties of the radial drawing. This could be done using the same way we introduce the difference metrics of dynamic hierarchical graphs presented in Section 4.3.
  - Combining some drawing aesthetic criteria (like minimizing edge crossings or minimizing the drawing area) with the dynamic radial drawing of hierarchical graphs, as the same way we expressed these problems in Section 4.4.
  - In addition to the previous points, practical user studies should be included in the problem of dynamic radial drawings of hierarchical graphs.

Bibliography

[AFT11] E. Ackerman, R. Fulek, and C. D. Tóth, *On the size of graphs that admit polyline drawings with few bends and crossing angles*, Rev. Selected Papers from the 18th Int. Sym. on Graph Drawing (GD'10), LNCS, vol. 6502, Springer-Verlag, 2011, pp. 1–12. 21

[AHN07] R. Andreev, P. Healy, and N.S. Nikolov, *Applying ant colony optimization metaheuristic to the dag layering problem*, IEEE Int. Sym. on Parallel & Distributed Processing (IPDPS'07), 2007, pp. 1–9. 43

[AHT02] D. Abelson, S.-H. Hong, and D. E. Taylor, *A group-theoretic method for drawing graphs symmetrically*, Rev. Papers from 10th Int. Sym. on Graph Drawing (GD'02), LNCS, vol. 2528, Springer-Verlag, 2002, pp. 86–97. 77

[AHU83] A. V. Aho, J. E. Hopcroft, and J. D. Ulman, *Data structure and algorithms*, Adison-Wiesley, Reading, MA, 1983. 48

[APP11a] D. Archambault, H. Purchase, and B. Pinaud, *Animation, small multiples, and the effect of mental map preservation in dynamic graphs*, IEEE Trans. Vis. Comput. Graph. **17** (2011), 539–552. 8, 76

[APP11b] D. Archambault, H. C. Purchase, and B. Pinaud, *Difference map readability for dynamic graphs*, Rev. Selected Papers from the 18th Int. Sym. on Graph Drawing (GD'10), LNCS, vol. 6502, Springer-Verlag, 2011, pp. 50–61. 74, 76

[Auy90] A. Auyyamathiti, *Computer assisted database design normalization and layout of entity-relationship diagrams*, Master's thesis, University of the Philippines, 1990. 3, 6

[Bac07] C. Bachmaier, *A radial adaptation of the sugiyama framework for visualizing hierarchical information*, IEEE Trans. Vis. Comput. Graph. **13** (2007), no. 3, 583 –594. 140

[Bac09] ———, *A generealized framework for drawing directed graphs*, Habilitation thesis, University of Passau, 2009. 7, 38, 41, 45, 50, 52, 56, 57

[BBBF12] C. Bachmaier, F. J. Brandenburg, W. Brunner, and R. Fülöp, *Drawing recurrent hierarchies*, JGAA **16** (2012), no. 2, 151–198. 7

[BBBH11] C. Bachmaier, F. J. Brandenburg, W. Brunner, and F. Hübner, *Global k-level crossing reduction*, JGAA **15** (2011), no. 5, 631–659. 7, 10, 43, 46, 53, 54, 55, 68, 70, 140

[BBFMM04] C. Bachmaier, F. J. Brandenburg, P. Forster M., Hilleis, and Raitner M., *Gravisto: Graph visualization toolkit*, Rev. Selected Papers from 12th Int. Sym. on Graph Drawing (GD'04), LNCS, vol. 3383, Springer-Verlag, 2004, pp. 502–503. 56, 59

[BBG11] C. Bachmaier, W. Brunner, and A. Gleissner, *Grid sifting: Leveling and crossing reduction.*, Technical report mip-1103, University of Passau, 2011. 7, 43, 67, 140

[BC87] N. S. Bhatt and S. S. Cosmadakis, *The complexity of minimizing wire lengths in vlsi layouts*, Inf. Process. Lett. **25** (1987), no. 4, 263 – 267. 3, 25

[BCD89] P. Benedusi, A. Cimitile, and U. De Carlini, *A reverse engineering methodology to reconstruct hierarchical data flow diagrams for software maintenance*, Proc. Conf. on Software Maintenance, 1989, pp. 180–189. 6

[BCMW04] P. Bose, J. Czyzowicz, P. Morin, and D. R. Wood, *The maximum number of edges in a three-dimensional grid-drawing*, JGAA **8** (2004), 21–26. 24

[BDD+10] C. Binucci, E. Di Giacomo, W. Didimo, A. Estrella-Balderrama, F. Frati, S. G. Kobourov, and G. Liotta, *Upward straight-line embeddings of directed graphs into point sets*, Comp. Geo. **43** (2010), no. 2, 219–232. 20

[BDLN05] C. Binucci, W. Didimo, G. Liotta, and M. Nonato, *Orthogonal drawings of graphs with vertex and edge labels*, Comp. Geo. **32** (2005), no. 2, 71–114. 22

[BFF05] C. Bachmaier, F. Fischer, and M. Forster, *Radial coordinate assignment for level graphs*, Computing and Combinatorics, LNCS, vol. 3595, Springer-Verlag, 2005, pp. 401–410. 140

[BFN85] C. Batini, L. Furlani, and E. Nardelli, *What is a good diagram? a pragmatic approach*, Proc. 4th Int. Conf. on Entity-Relationship Approach, IEEE Computer Society, 1985, pp. 312–319. 27

[BGHM07] A. Barsky, J. L. Gardy, R. E. W. Hancock, and T. Munzner, *Cerebral: a cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation*, Bioinformatics **23** (2007), no. 8, 1040–1042. 3

[Bie96] T. Biedl, *Optimal orthogonal drawings of triconnected plane graphs*, Proc. 5th Scandinavian Workshop on Algorithm Theory, Springer-Verlag, 1996, pp. 333–344. 22

[Bie98] ———, *New lower bounds for orthogonal graph drawings*, JGAA **2** (1998), 1–31. 22

[Bie11] ———, *Drawing some planar graphs with integer edge-lengths*, CCCG'11, 2011, http://www.cccg.ca/Proc./2011/papers/paper36.pdf. 20

[BJ03] C. Buchheim and M. Jünger, *Detecting symmetries by branch & cut*, Mathematical Programming **98** (2003), 369–384. 77

[BJG08] J. Bang-Jensen and G. Z. Gutin, *Digraphs: Theory, algorithms and applications*, 2nd ed., Springer Publishing Company, Incorporated, 2008. 37, 41

[BJL01] C. Buchheim, M. Jünger, and S. Leipert, *A fast layout algorithm for k-level graphs*, Proc. 8th Int. Sym. on Graph Drawing (GD'00), LNCS, vol. 1984, Springer-Verlag, 2001, pp. 229–240. 60, 67

[BK94] T. Biedl and G. Kant, *A better heuristic for orthogonal graph drawings*, Algorithms ESA'94, LNCS, vol. 855, Springer-Verlag, 1994, pp. 24–35. 22

[BK02] C. Borgelt and R. Kruse, *Graphical models: methods for data analysis and mining*, J. Wiley, 2002. 2, 60, 67

[BKRW11] T. Bläsius, M. Krug, I. Rutter, and D. Wagner, *Orthogonal graph drawing with flexibility constraints*, Rev. Selected Papers from the 18th Int. Sym. on Graph Drawing (GD'10), LNCS, vol. 6502, Springer-Verlag, 2011, pp. 92–104. 22

[BKS10] M. Bayati, J. H. Kim, and A. Saberi, *A sequential algorithm for generating random graphs*, Algorithmica **58** (2010), no. 4, 860–910. 140

[BL76] K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms*, Jou. comp. Syst. Sci. **13** (1976), 335–379. 20

[BL84] S. N. Bhatt and F. T. Leighton, *A framework for solving vlsi graph layout problems*, Jou. comp. Syst. Sci. **28** (1984), no. 2, 300–343. 5

[BL10] U. Brandes and J. Lerner, *Structural similarity: Spectral methods for relaxed blockmodeling*, Journal of Classification **27** (2010), 279–306. 27, 77

[BLME02] J. Branke, S. Leppert, M. Middendorf, and P. Eades, *Width-restricted layering of acyclic digraphs with consideration of dummy nodes*, Information Processing Letter **81** (2002), 59–63. 43

[BM01] O. Bastert and C. Matuszweski, *Layered drawings of digraphs*, Drawing graphs: Methods and Models, vol. 2025, Springer-Verlag, 2001, pp. 228–246. 37

[BM12] U. Brandes and M. Mader, *A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing*, Rev. Selected Papers from the 19th Int. Sym. on Graph Drawing (GD'11), LNCS, vol. 7034, Springer-Verlag, 2012, pp. 99–110. 74

[BMNR10] S. Biswas, D. Mondal, R. I. Nishat, and M. S. Rahman, *Minimum-segment convex drawings of 3-connected cubic plane graphs*, Proc. of 16th Annual Int. Conf. on Computing and Combinatorics (COCOON'10), LNCS, vol. 6196, Springer-Verlag, 2010, pp. 182–191. 22

[BP90] K.-F. Böhringer and F. N. Paulisch, *Using constraints to achieve stability in automatic graph layout algorithms*, Proc. SIGCHI Conf. on Human factors in computing systems: Empowering people, CHI'90, ACM, 1990, pp. 43–51. 77, 82

[Bra01] J. Branke, *Dynamic graph drawing*, Drawing graphs: Methods and Models, vol. 2025, Springer-Verlag, 2001, pp. 228–246. 74, 77, 82, 140

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified modeling language user guide*, Adison Wiesley Longman, 1999. 3

[BS88] J. Bhasker and S. Sahni, *A linear algorithm to find a rectangular dual of a planar triangulated graph*, Algorithmica **3** (1988), 247–278. 23

[BS90] B. Berger and P. W. Shor, *Approximation alogorithms for the maximum acyclic subgraph problem*, Proc. 1st Annual ACM-SIAM Sym. on Discrete Algorithms, SODA'90, Society for Industrial and Applied Mathematics, 1990, pp. 236–243. 37, 39

[BT00] S. S. Bridgeman and R. Tamassia, *Difference metrics for interactive orthogonal graph drawing algorithms*, JGAA **4** (2000), no. 3, 47–74. 9, 10, 74, 76, 78, 80, 81, 88, 89, 110, 139

[BT01] Stina S. Bridgeman and Roberto Tamassia, *A user study in similarity measures for graph drawing*, Proc. 8th Int. Sym. on Graph Drawing (GD'00), LNCS, vol. 1984, Springer-Verlag, 2001, pp. 19–30. 9, 27, 76, 77

[BTT84] C. Batini, M. Talamo, and R. Tamassia, *Computer aided layout of entity relationship diagrams*, Jou. of System. and Soft. **4** (1984), no. 2-3, 163–173. 3, 6

[BVB+11] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf, *Parallel edge splatting for scalable dynamic graph visualization*, IEEE Trans. Vis. Comput. Graph. **17** (2011), no. 12, 2344 –2353. 74

[BW97] U. Brandes and D. Wagner, *A bayesian paradigm for dynamic graph layout*, Proc. 5th Int. Sym. on Graph Drawing (GD'97), LNCS, vol. 1353, Springer-Verlag, 1997, pp. 236–247. 74

[BW98] ———, *Dynamic grid embedding with few bends and changes*, Algorithms and Computation, LNCS, vol. 1533, Springer-Verlag, 1998, pp. 90–99. 24

[Car80] M.-J. Carpano, *Automatic display of hierarchized graphs for computer-aided decision analysis*, IEEE Trans. Sys. Man & Cyber. **10** (1980), no. 11, 705–715. 7, 34, 70

[Cat95] T. Catarci, *The assignment heuristic for crossing reduction*, IEEE Trans. Sys. Man & Cyber. **25** (1995), no. 3, 515–521. 43

[CDT+92] R. F. Cohen, G. Di Battista, R. Tamassia, I. G. Tollis, and P. Bertolazzi, *A framework for dynamic graph drawing*, Proc. 8th annual Sym. on Comp. Geo., SCG'92, ACM, 1992, pp. 261–270. 74

[CDTT95] R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis, *Dynamic graph drawings: Trees, series-parallel digraphs, and planar st-digraphs*, SIAM Journal Computing **24** (1995), no. 5, 970–1001. 81

[CG72] E. G. Coffman and R. L. Graham, *Optimal scheduling for two-processor systems*, Acta Informatica **1** (1972), 200–213. 40, 41

[CG12] M. Chimani and C. Gutwenger, *Advances in the planarization method: Effective multiple edge insertions*, Graph Drawing, LNCS, vol. 7034, Springer-Verlag, 2012, pp. 87–98. 20

[CGH+97] L. P. Chew, . T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, and D. Kravets, *Geometric pattern matching under euclidean motion*, Comp. Geo. - Theory and Applications **7** (1997), 113–124. 80

[CGMW10] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong, *Layer-free upward crossing minimization*, ACM Journal of Experimental Algorithmics **15** (2010), 2.2:2.1–2.2:2.27. 43, 68

[CGMW11] ———, *Upward planarization layout*, JGAA **15** (2011), no. 1, 127–155. 43, 67, 140

[CGT96] M. Chrobak, M. T. Goodrich, and R. Tamassia, *Convex drawings of graphs in two and three dimensions (preliminary version)*, Proc. 12th annual Sym. on Comp. Geo. (SCG'96) (New York, NY, USA), ACM, 1996, pp. 319–328. 22

[Cha11] *McCort enterprise company organization chart*, http://itsabout.server304.com/wp-content/uploads/2011/06/chart.jpg, June, 2011. 7

[Che76] P.-S. Chen, *The entity-relationship model—toward a unified view of data*, ACM Transaction on Database Systems **1** (1976), no. 1, 9–36. 3, 6

[CHT89] J. Cai, X. Han, and R. Tarjan, *New solutions for planar graph problems*, Tech. report, Department of Computer Science, New York University / Courant Institute, 1989. 20

[CKN+03] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler, *A system for graph-based visualization of the evolution of software*, Proc. ACM Sym. on Software visualization 2003 (New York, NY, USA), SoftVis '03, ACM, 2003, pp. 77–ff. 3

[CM08] M. Chein and M.-L. Mugnier, *Graph-based knowledge representation: Computational foundations of conceptual graphs*, 1 ed., Springer Publishing Company, Incorporated, 2008. 2

[CM11] S. Cabello and B. Mohar, *Crossing number and weighted crossing number of near-planar graphs*, Algorithmica **60** (2011), no. 3, 484–504. 44

[CN98] M. Chrobak and S.-I. Nakano, *Minimum-width grid drawings of plane graphs*, Comp. Geo. **11** (1998), no. 1, 29–54. 24

[CNAO85] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa, *A linear algorithm for embedding planar graphs using pq-trees*, Jou. comp. Syst. Sci. **30** (1985), 54–76. 20

[CR06] Y. Cheung and S. Rowlinson, *Relationship management - drawing on international experiences*, http://eprints.qut.edu.au/17868/, May 2006. 6

[Cru93] I. F. Cruz, *User-defined visual languages for querying data*, Tech. report, Providence, RI, USA, 1993. 6

[CY02] M.-C. Chuang and H.-C. Yen, *On nearly symmetric drawings of graphs*, Proc. 6th Int. Conf. on Information Visualisation, 2002, pp. 489 – 494. 77

[CYN84] N. Chiba, T. Yamanouchi, and T. Nishizeki, *Linear algorithm for convex drawing of planar graphs*, Progress in graph theory, Academic Press, 1984, pp. 153–173. 22

[DDF+06] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, *A survey of autonomic communications*, ACM Trans. Auton. Adapt. Syst. **1** (2006), no. 2, 223–259. 8

[DDL07] E. Di Giacomo, W. Didimo, and G. Liotta, *Radial drawings of graphs: geometric constraints and trade-offs*, Rev. Papers 14th Int. Sym. on Graph drawing (GD'06), LNCS, vol. 4372, Springer-Verlag, 2007, pp. 355–366. 140

[Deb01] K. Deb, *Multi-objective optimization using evolutionary algorithms*, Wiley-Interscience Series in Systems and Optimization, John Wiley & Sons, Chichester, 2001. 77, 140

[DEG+11] C. A. Duncan, D. Eppstein, M. T. Goodrich, S. G. Kobourov, and M. Nöllenburg, *Drawing trees with perfect angular resolution and polynomial area*, Rev. Selected Papers from the 18th Int. Sym. on Graph Drawing (GD'10), LNCS, vol. 6502, Springer-Verlag, 2011, pp. 183–194. 30

[DETT94] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Algorithms for drawing graphs: an annotated bibliography*, Comp. Geo. Theory and Applications **4** (1994), 235–282. 3, 5, 20, 31

[DETT99] ———, *Graph drawing: Algorithms for the visualization of graphs*, Prentice Hall, New Jersey, USA, 1999. 3, 26, 31, 33, 46, 49, 52

[DF07] Y. Donoso and R. Fabregat, *Multi-objective optimization in computer networks using metaheuristics*, Auerbach Publications, Boston, MA, USA, 2007. 77, 140

[DG02] S. Diehl and C. Görg, *Graphs, they are changing*, Rev. Papers from 10th Int. Sym. on Graph Drawing (GD'02), LNCS, vol. 2528, Springer-Verlag, 2002, pp. 23–30. 75

[DHW11] C. Doll, T. Hartmann, and D. Wagner, *Fully-dynamic hierarchical graph clustering using cut trees*, Proc. 12th Int. Conf. on Algorithms and data structures, WADS'11, Springer-Verlag, 2011, pp. 338–349. 7, 74

[DKL+12] G. Di Battista, E. Kim, G. Liotta, A. Lubiw, and S. Whitesides, *The shape of orthogonal cycles in three dimensions*, Discrete & Comp. Geo. **47** (2012), no. 3, 461–491. 22

[DLF+09] T. Dwyer, B. Lee, D. Fisher, K. I. Quinn, P. Isenberg, G. Robertson, and C. North, *A comparison of user-generated and automatic graph layouts*, IEEE Trans. Vis. Comput. Graph. **15** (2009), no. 6, 961–968. 9, 27, 75, 76, 113

[DPS11] T. Dey, D. J. Phillips, and P. Steele, *A graphical tool to visualize predicted minimum delay flights*, JCGS **20** (2011), no. 2, 294–297. 2

[DT89] G. Di Battista and R. Tamassia, *Incremental planarity testing*, Proc. 30th Annual Sym. on Foundations of Computer Science, IEEE Computer Society, 1989, pp. 436–441. 81

[Ead84] Peter Eades, *A heuristic for graph drawing*, Congressus Numerantium **42** (1984), 149–160. 107

[Ead88] P. Eades, *Symmetry finding algorithms*, Computational Morphology (G. T. Toussaint, Ed.), 1988, pp. 41–51. 27

[Ead08] _____, *Some constrained notions of planarity*, Proc. 19th Int. Sym. on Algorithms and Computation (ISAAC 2008), LNCS, vol. 5369, Springer-Verlag, 2008, pp. 2–2. 20

[EFL97] P. Eades, Q.-W. Feng, and X. Lin, *Straight-line drawing algorithms for hierarchical graphs and clustered graphs*, Proc. Sym. on Graph Drawing (GD'96), LNCS, vol. 1190, Springer-Verlag, 1997, pp. 113–128. 5, 7

[EFN06] P. Eades, X. Feng, Q. andLin, and H. Nagamochi, *Straight-line drawing algorithms for hierarchical graphs and clustered graphs*, Algorithmica **44** (2006), no. 1, 1–32. 7, 57, 66

[EGDB02] T. Eschbach, W. Günther, R. Drechsler, and B. Becker, *Crossing reduction by windows optimization*, Rev. Papers from 10th Int. Sym. on Graph Drawing (GD'02), LNCS, vol. 2528, Springer-Verlag, 2002, pp. 285–294. 43, 57

[EGK+03] j. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, *Graphviz and dynagraph - static and dynamic graph drawing tools*, Graph Drawing Software, Springer-Verlag, 2003, pp. 127–148. 74

[EK86] P. Eades and D. Kelly, *Heuristics for reducing crossings in 2-layered networks*, Ars Combinatorica **21** (1986), no. A, 89–98. 43, 46, 50

[EKLN04] C. Erten, S. G. Kobourov, V. Le, and A. Navabi, *Simultaneous graph drawing: Layout algorithms and visualization schemes*, Rev. Papers from 11th Int. Sym. on Graph Drawing (GD'03), LNCS, vol. 2912, Springer-Verlag, 2004, pp. 437–449. 74

[EL93] P. Eades and T. Lin, *Algorithmic and declarative approach to aesthetic layout*, ALCOM Int. Workshop on Graph Drawing and Topological Algorithms (GD'93), 1993, pp. 62–63. 26

[Elm77] S. E. Elmaghraby, *Activity networks: project planning and control by network models*, Wiley-Interscience publication, Wiley, New York, NY, USA, 1977. 2, 6

[ELMN11] D. Eppstein, M. Löffler, E. Mumford, and M. Nöllenburg, *Optimal 3d angular resolution for low-degree graphs*, Rev. Selected Papers from the 18th Int. Sym. on Graph Drawing (GD'10), LNCS, vol. 6502, Springer-Verlag, 2011, pp. 208–219. 30

[ELMS91] P. Eades, W. Lai, K. Misue, and K. Sugiyama, *Preserving the mental map of a diagram*, Proc. of COMPUGRAPHICS (1991), vol. 91, 1991, pp. 24–33. 8, 74, 75, 78, 79, 80

[ELS93] P. Eades, X. Lin, and W. F. Smyth, *A fast and effective heuristic for the feedback arc set problem*, Information Processiing Letters **47** (1993), 319–323. 38

[ELT96] P. Eades, X. Lin, and R. Tamassia, *An algorithm for drawing a hierarchical graph*, Int. J. Comp. Geo. Appl. **6** (1996), no. 2, 145–156. 5, 7, 53, 54, 57, 60

[ENRS00] G. Even, J. Naor, S. Rao, and B. Schieber, *Divide-and-conquer approximation algorithms via spreading metrics*, JACM **47** (2000), 585–616. 39

[ES91] P. Eades and K. Sugiyama, *How to draw a directed graph*, Journal of Information Processing **13** (1991), 424–437. 34, 41, 53, 60, 70

[ESW00] P. Eades, A. Symvonis, and S. Whitesides, *Three-dimensional orthogonal graph drawing algorithms*, Disc. Appl. Math. **103** (2000), no. 1-3, 55–87. 22

[Eul36] L. Euler, *Solutio problematis ad geometriam situs pertinentis*, Commentarii Academiae Scientiarum Imperialis Petropolitanae **8** (1736), 128–140. 3

[EW94a] P. Eades and S. Whitesides, *Drawing graphs in two layers*, Theoretical Computer Science **131** (1994), 361–374. 43, 44, 48, 49

[EW94b] P. Eades and N. C. Wormald, *Edge crossings in drawings of bipartite graphs*, Algorithmica **11** (1994), no. 4, 379–403. 43, 44, 46

[Far48] I. Fary, *On straight line representations of planar graphs*, Acta Scientiarum Mathematicarum **11** (1948), 229–233. 20

[Fen97] Q. Feng, *Algorithms for drawing clustered graphs*, Ph.D. thesis, Department of Computer Science and Software Engineering, The University of Newcastle, Australia, 1997. 5, 19

[FH01] R. Fleischer and C. Hirsch, *Graph drawing and its applications*, Drawing graphs: Methods and Models, Springer-Verlag, 2001, pp. 1–22. 31

[Flo] *Flow-graphs from bound-t analysis of sparc/erc32 example*, `http://www.bound-t.com/targets/sparc/example-brochure/dot-code/fg_exa__count_007.png`. 7

[FPP90] H. Fraysseix, J. Pach, and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica **10** (1990), no. 1, 41–51. 24

[FQ11] M. Farrugia and A. J. Quigley, *Effective temporal graph layout: A comparative study of animation versus static display methods.*, Information Visualization **10** (2011), no. 1, 47–64. 76

[Fri97] A. Frick, *Upper bounds on the number of hidden nodes in sugiyama's algorithm*, Proc. Sym. on Graph Drawing (GD'96), LNCS, vol. 1190, Springer-Verlag, 1997, pp. 169–183. 7

[FT04] Y. Frishman and A. Tal, *Dynamic drawing of clustered graphs*, IEEE Sym. on Information Visualization, INFOVIS'2004, 2004, pp. 191–198. 74, 81

[Gan03] E. R. Gansner, *Drawing graphs with GraphViz*, Tech. report, AT&T Bell Laboratories, Murray Hill, NJ, USA, 2003. 6

[Gan07] ———, *Drawing graphs with graphviz*, Library (2007), no. August. 6

[GBPD04] C. Görg, P. Birke, M. Pohl, and S. Diehl, *Dynamic graph drawing of sequences of orthogonal and hierarchical graphs*, Rev. Selected Papers from 12th Int. Sym. on Graph Drawing (GD'04), LNCS, vol. 3383, Springer-Verlag, 2004, pp. 228–238. 7, 74, 75

[GHK10] E. R. Gansner, Y. Hu, and S. Kobourov, *Gmap: Drawing graphs as maps*, Rev. Papers 17th Int. Sym. on Graph Drawing (GD'09), LNCS, vol. 5849, 2010, pp. 405–407. 2

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of np-completeness (series of books in the mathematical sciences)*, W. H. Freeman & Co., New York, USA, 1979. 37, 40, 41

[GJ83] ———, *Crossing number is NP-complete*, SIAM Journal on Algebraic and Discrete Methods **4** (1983), no. 3, 312–316. 44

[GJR85] M. Grötschel, M. Jünger, and G. Reinelt, *On the acyclic subgraph polytope*, Mathematical Programming **33** (1985), 28–42. 39

[GKNV93] E. R. Gansner, E. Koutsofios, S. C. North, and K. Vo, *A technique for drawing directed graphs*, IEEE Trans. Soft. Eng. **19** (1993), no. 3, 214–230. 34, 39, 41, 43, 49, 66, 70

[GM89] D. J. Gschwind and T. P. Murtagh, *A recursive algorithm for drawing hierarchical directed graphs*, Technical report cs-89-02, Department of Computer Science, Williams College, 1989. 7, 34, 70

[GM98] C. Gutwenger and P. Mutzel, *Planar polyline drawings with good angular resolution*, Proc. 6th Int. Sym. on Graph Drawing (GD'98), LNCS, vol. 1547, Springer-Verlag, 1998, pp. 167–182. 21

[GMH+06] Amy L Griffin, Alan M MacEachren, Frank Hardisty, Erik Steiner, and Bonan Li, *A comparison of animated maps with static small-multiple maps for visually identifying space-time clusters*, Annals of the Association of American Geographers **96** (2006), no. 4, 740–753. 76

[GMO99] M. T. Goodrich, J. S. B. Mitchell, and M. W. Orletsky, *Approximate geometric pattern matching under rigid motions*, IEEE Trans. Pattern Anal. Mach. Intell. **21** (1999), no. 4, 371–379. 80

[GMS82] A. Grinvald, A. Manker, and M. Segal, *Visualization of the spread of electrical activity in rat hippocampal slices by voltage-sensitive optical probes*, The Journal of Physiology **333** (1982), 269–291. 2

[GMZ09] C. Gutwenger, P. Mutzel, and B. Zey, *On the hardness and approximability of planar biconnectivity augmentation*, Proc. 15th Annual Int. Conf. on Computing and Combinatorics (COCOON'9), LNCS, vol. 5609, Springer-Verlag, 2009, pp. 249–257. 20

[GNV88] E. R. Gansner, S. C. North, and K. P. Vo, *Dag - a program that draws directed graphs*, Software: Practice and Experience **18** (1988), 1047–1062. 34, 60, 70

[GP83] E. J. Goodman and R. Pollack, *Multidimensional sorting*, SIAM J. on Computing **12** (1983), 484–507. 44, 79

[GR07] A. Garg and A. Rusu, *Area-efficient planar straight-line drawings of outerplanar graphs*, Disc. Appl. Math. **155** (2007), no. 9, 1116–1140. 20

[Gro08] J. L. Gross, *Combinatorial methods with computer applications*, Discrete mathematics and its applications, Chapman & Hall/CRC, 2008. 1

[GSBM01] W. Günther, R. Schönfeld, B. Becker, and P. Molitor, *k-layer straightline crossing minimization by speeding up sifting*, Proc. 8th Int. Sym. on Graph Drawing (GD'00), LNCS, vol. 1984, Springer-Verlag, 2001, pp. 253–258. 43

[GT02] A. Garg and R. Tamassia, *On the computational complexity of upward and rectilinear planarity testing*, SIAM J. on Computing **31** (2002), no. 2, 601–625. 3

[GW06] M. Gaertler and D. Wagner, *A hybrid model for drawing dynamic and evolving graphs*, Algorithmic Aspects of Large and Complex Networks, Dagstuhl Seminar Proc., no. 05361, Internationales Begegnungs-und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. 74

[GY99] J. L. Gross and J. Yellen, *Graph theory and its applications*, CRC Press, Inc., Boca Raton, FL, USA, 1999. 1

[HA99a] B. A. Huberman and L. A. Adamic, *Evolutionary dynamics of the world wide web*, 1999, http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/9901071. 2, 74

[HA99b] _____, *Growth dynamics of the world-wide web*, Nature **401** (1999), http://dx.doi.org/10.1038/43604. 74

[HE03] S.-H. Hong and P. Eades, *Drawing trees symmetrically in three dimensions*, Algorithmica **36** (2003), no. 2, 153–178. 27

[HEL05] X. Huang, P. Eades, and W. Lai, *A framework of filtering, clustering and dynamic layout graphs for visualization*, Proc. 28th Australasian Conf. on Computer Science, ACSC'05, vol. 38, Australian Computer Society, Inc., 2005, pp. 87–96. 74

[HGK10] Y. Hu, E. R. Gansner, and S. Kobourov, *Gmap: Visualizing graphs and clusters as maps*, IEEE Computer Graphics and Applications **30** (2010), no. 6, 54–66. 2

[HL06] X. Huanga and W. Laib, *Clustering graphs for visualization via node similarities*, J. Vis. Lang. Comput. **17** (2006), 125 – 253. 27, 77

[HN02] P. Healy and N. S. Nikolov, *How to layer a directed acyclic graph*, Rev. Papers from 9th Int. Sym. on Graph Drawing (GD'01), LNCS, vol. 2265, Springer-Verlag, 2002, pp. 16–30. 42

[HS07] P. Hlinený and G. Salazar, *On the crossing number of almost planar graphs*, Rev. Papers 14th Int. Sym. on Graph drawing (GD'06), LNCS, vol. 4372, Springer-Verlag, 2007, pp. 162–173. 44

[HS12] I. Halupczok and A. Schulz, *Pinning paloons with perfect angles and optimal area*, Rev. Selected Papers from the 19th Int. Sym. on Graph Drawing (GD'11), LNCS, vol. 7034, Springer-Verlag, 2012, pp. 154–165. 30

[HT74] J. Hopcroft and R. Tarjan, *Efficient planarity testing*, JACM **21** (1974), 549–568. 20

[Hua07] W. Huang, *Using eye tracking to investigate graph layout effects*, 6th Int. Asia-Pacific Sym. on Visualization, APVIS'07, 2007, pp. 97–100. 76

[HvW09] D. Holten and J. J. van Wijk, *A user study on visualizing directed edges in graphs*, Proc. 27th Int. Conf. on Human factors in computing systems, CHI'09, ACM, 2009, pp. 2299–2308. 9, 27, 76

[IRW99] E. Ihler, G. Reich, and P. Widmayer, *Class steiner trees and vlsi-design*, Disc. Appl. Math. **90** (1999), no. 1-3, 173–194. 25

[ISI89] K. Imai, S. Sumino, and H. Imai, *Minimax geometric fitting of two corresponding sets of points*, Proc. 5th annual Sym. on Comp. Geo., SCG'89, ACM, 1989, pp. 206–275. 80

[Ism04] A. A. K. Ismaeel, *A study of some akgorithms for drawing level graphs*, Master's thesis, Computer Science Department, Minia University, Egypt, 2004. 7, 60

[JG09] Z. Jiang and S. K. Gupta, *Threshold testing: Improving yield for nanoscale vlsi*, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **28** (2009), no. 12, 1883–1895. 3, 25

[JK07] S. Jenkins and S. R. Kirk, *Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution*, JIS **177** (2007), no. 12, 2587–2601. 6

[JLMO97] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal, *A polyhedral approach to the multi-layer crossing minimization problem*, Proc. 5th Int. Sym. on Graph Drawing (GD'97), LNCS, vol. 1353, Springer-Verlag, 1997, pp. 13–24. 43, 56

[JM97] M. Jünger and P. Mutzel, *2-layer straightline crossing minimization: performance of exact and heuristic algorithms*, JGAA **1** (1997), 1–25. 43, 47, 51, 52, 53

[JM04] ———, *Graph drawing software*, Series on Mathematics and Visualization, vol. 12, 2004. 3, 4, 6

[Kam89] T. Kamada, *Visualizing abstract objects and relations - a constrained-based approach*, Series in Computer Science, vol. 5, World Scientific, Singapore, 1989. 3

[Kan93] G. Kant, *Algorithms for drawing planar graphs*, Ph.D. thesis, Faculteit Wiskunde en Informatica, Utrecht University, The Netherlands, 1993. 22

[Kar72] R. M. Karp, *Reducibility among combinatorial problems*, Proc. of a Sym. on the Complexity of Computer Computations, The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 85–103. 37

[Kar09] M. R. Karim, *Straight-line grid drawings of planar graphs: with sub-quadratic area*, VDM Verlag Dr. Müller, 2009. 20, 24

[KG06] G. Kumar and M. Garland, *Visual exploration of complex time-varying graphs*, IEEE Trans. Vis. Comput. Graph. **12** (2006), no. 5, 805–812. 74, 140

[KHI+03] R. Kosara, C. G. Healey, V. Interrante, D. H. Laidlaw, and C. Ware, *User studies: Why, how, and when?*, IEEE Computer Graphics and Applications **23** (2003), no. 4, 20–25. 114

[KLTT97] G. Kant, G. Liotta, R. Tamassia, and I. G. Tollis, *Area requirement of visibility representations of trees*, Inf. Process. Lett. **62** (1997), no. 2, 81–88. 24

[KPL06] P. Kuntz, B. Pinaud, and R. Lehn, *Minimizing crossings in hierarchical digraphs with a hybridized genetic algorithm*, Journal of Heuristics **12** (2006), 23–36. 7, 43, 57

[KT12] E. M. Kornaropoulos and I. G. Tollis, *Overloaded orthogonal drawings*, Proc. 19th Int. Sym. on Graph Drawing (GD 2011), LNCS, vol. 7034, Springer-Verlag, 2012, pp. 242–253. 22

[KW01] Michael Kaufmann and Dorothea Wagner, *Drawing graphs: Methods and models*, LNCS, vol. 2025, Springer-Verlag, London, UK, 2001. 3, 33, 45, 46, 47, 53, 66

[LE99] X. Lin and P. Eades, *Area minimization for grid visibility representation of hierarchically planar graphs*, Proc. of 5th Annual Int. Conf. on Computing and Combinatorics (COCOON'99), LNCS, vol. 1627, Springer-Verlag, 1999, pp. 92–102. 24

[Lei98] S. Leipert, *Level planarity testing and embedding in linear time*, Ph.D. thesis, Computer Science Department, University of Cologne, 1998. 7

[Lin92] X. Lin, *Analysis of algorithms for drawing graphs*, Ph.D. thesis, Department of Computer Science, University of Queensland, Australia, 1992. 42

[LLY06] Y.-Y. Lee, C.-C. Lin, and H.-C. Yen, *Mental map preserving graph drawing using simulated annealing*, Proc. Asia-Pacific Sym. on Information Visualisation 2006, APVis'06, vol. 60, Australian Computer Society, Inc., 2006, pp. 179–188. 8

[LM99] M. Laguna and R. Martí, *Grasp and path relinking for 2-layer straight line crossing minimization*, INFORMS Journal on Computing **11** (1999), 44–52. 43, 51

[LMR98] K. A. Lyons, H. Meijer, and D. Rappaport, *Algorithms for cluster busting in anchored graph drawing*, JGAA **1** (1998), no. 1, 1–24. 74, 78, 79, 88

[LMV97] M. Laguna, R. Martí, and Valls V., *Arc crossing minimization in hierarchical digraphs with tabu search*, Computers and Operations Research **24** (1997), no. 12, 1175–1186. 43, 51, 57

[LNS85] R. J. Lipton, S. C. North, and J. S. Sandberg, *A method for drawing graphs*, Proc. 1rst annual Sym. on Comp. Geo., SCG'85, ACM, 1985, pp. 153–160. 27

[LS77] S. Lam and R. Sethi, *Worst case analysis of two scheduling algorithms*, SIAM J. on Computing **6** (1977), no. 3, 518–536. 41

[LY05] C.-C. Lin and H.-C. Yen, *A new force-directed graph drawing method based on edge-edge repulsion*, Proc. 9th Int. Conf. on Information Visualisation, IEEE Computer Society, 2005, pp. 329–334. 30

[Lyo92] K. A. Lyons, *Cluster busting in anchored graph drawing*, Proc. Conf. of the Centre for Advanced Studies on Collaborative research 1992, CASCON '92, vol. 1, IBM Press, 1992, pp. 7–17. 78

[Mäk90] E. Mäkinen, *Experiments on drawing 2-level hierarchical graphs*, IJCM **37** (1990), no. 3-4, 129–135. 49, 50

[Man91] J. Manning, *Computational complexity of geometric symmetry detection in graphs*, Proc. 1st Great Lakes Computer Science Conf. on Computing in the 90's, Springer-Verlag, 1991, pp. 1–7. 77

[MDV06] V. Manohararajah, S. B. Dean, and Z. G. Vranesic, *Heuristics for area minimization in lut-based fpga technology mapping*, IEEE Transactions on CAD of Integrated Circuits and Systems **25** (2006), no. 11, 2331–2340. 28

[Meh84] K. Mehlhorn, *Data structures and algorithms. volume 2: Graph algorithms and np-completeness.*, EATCS Monographs on Theoretical Computer Science, Springer, 1984. 41

[MELS95] K. Misue, P. Eades, W. Lai, and K. Sugiyama, *Layout adjustment and the mental map*, J. Vis. Lang. Comput. **6** (1995), no. 2, 183 – 210. 8, 74, 75, 78, 79, 80

[Men96] A. O. Mendelzon, *Visualizing the world wide web*, Proc. Workshop on Advanced visual interfaces AVI'96, ACM Press, 1996, pp. 13–19. 2

[Mes88] E. B. Messinger, *Automatic layout of large directed graphs*, Ph.D. thesis, University of Washington, 1988. 34, 70

[MHN05] K. Miura, H. Haga, and T. Nishizeki, *Inner rectangular drawings of plane graphs (extended abstract)*, Algorithms and Computation, LNCS, vol. 3341, Springer-Verlag, 2005, pp. 449–478. 23

[MHT93] K. Miriyala, S.W. Hornick, and R. Tamassia, *An incremental approach to aesthetic graph layout*, Proc. 6th Int. Workshop on Computer-Aided Software Engineering, CASE'93, july 1993, pp. 297–308. 81

[MMB06] C. Muelder, K.-L. Ma, and T. Bartoletti, *Interactive visualization for network and port scan detection*, 8th Int. Sym. of Recent Advances in Intrusion Detection, RAID 2005, LNCS, vol. 3858, Springer-Verlag, 2006, pp. 265–283. 2

[Moe90] S. Moen, *Drawing dynamic trees*, IEEE Software **7** (1990), no. 4, 21–28. 74, 81

[MR10] O. Macindoe and W. Richards, *Graph comparison using fine structure analysis*, Proc. IEEE Second Int. Conf. on Social Computing 2010 (Washington, DC, USA), SOCIALCOM '10, IEEE Computer Society, 2010, pp. 193–200. 8, 74

[MRH91] E. B. Messinger, L. A. Rowe, and R.H. Henry, *A divide-and-conquer algorithm for the automatic layout of large directed graphs*, IEEE Trans. Sys. Man & Cyber. **21** (1991), no. 1, 1–2. 34, 70

[MS94] E. Mäkinen, , and M. Sieranta, *Genetic algorithms for drawing bipartite graphs*, IJCM **53** (1994), no. 3-4, 157–166. 51

[MSM99] C. Matuszewski, R. Schzönfeld, and P. Molitor, *Using sifting for k-layer straightline crossing minimization*, Proc. 7th Int. Sym. on Graph Drawing (GD'99), LNCS, vol. 1731, 1999, pp. 217–224. 43, 50, 54

[Mur09] P. Murrell, *Drawing diagrams with r*, The R Journal **1** (2009), no. 1, 15–21. 3

[Mut92] P. Mutzel, *A fast $\mathcal{O}(n)$ linear time embedding algorithm based on the hopcroft-tarjan planarity test*, Technical report 92.107, Köln University, 1992. 20

[Mut00] ———, *An alternative method to crossing minimization on hierarchical graphs*, SIAM Journal on Optimization **11** (2000), 1065–1080. 7, 43

[MUV02] X. Munoz, W. Unger, and I. Vrt'o, *One sided crossing minimization is np-hard for sparse graphs*, Rev. Papers from 9th Int. Sym. on Graph Drawing (GD'01), LNCS, vol. 2265, Springer-Verlag, 2002, pp. 115–122. 43, 44

[MV93] A. Marzal and E. Vidal, *Computation of normalized edit distance and applications*, IEEE Trans. Pattern Anal. Mach. Intell. **15** (1993), no. 9, 926 –932. 80

[Mye03] C. R. Myers, *Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs*, Phys. Rev. E **68** (2003), 046116–1–046116–15. 6

[Nis07] T Nishizeki, *Inner rectangular drawings of plane graphs: Application of graph drawing to vlsi layouts*, Proc. of First Workshop on Algorithms and Computation WALCOM'07, Bangladesh Academy of Sciences (BAS), 2007, pp. 1–2. 23, 25

[Nor97] S. North, *Incremental layout in dynadag*, Proc. Sym. on Graph Drawing (GD'96), LNCS, vol. 1190, Springer-Verlag, 1997, pp. 409–418. 74, 79, 82, 83

[NR04] T. Nishizeki and M. S. Rahman, *Planar graph drawing*, Lecture Notes Series on Computing, vol. 12, Elsevier Science Publishers B.V., Amstrdam, The Netherlands, 2004. 3, 31, 33

[NT06] N. S. Nikolov and A. Tarassov, *Graph layering by promotion of nodes*, Disc. Appl. Math. **154** (2006), 848–860. 42, 43

[NTB05] N. S. Nikolov, A. Tarassov, and J. Branke, *In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes*, ACM Journal of Experimental Algorithmics **10** (2005). 43

[NW88] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*, Wiley-Interscience, 1988. 42

[NW02] S. North and G. Woodhull, *Online hierarchical graph drawing*, Rev. Papers from 9th Int. Sym. on Graph Drawing (GD'01), LNCS, vol. 2265, Springer-Verlag, 2002, pp. 77–81. 7, 82

[NW11] M. Nollenburg and A. Wolff, *Drawing and labeling high-quality metro maps by mixed-integer programming*, IEEE Trans. Vis. Comput. Graph. **17** (2011), 626–641. 3

[Ost96] D. I. Ostry, *Some three-dimensional graph drawing algorithms*, Master's thesis, Department of Computer Science and Software Engineering, The University of New Castle, Australia, 1996. 26

[OW78] R. Otten and J. van Wijk, *Graph representations in interactive layout design*, Proc. IEEE Int. Sym. on Circuits and Systems, 1978, pp. 914–918. 24, 74

[PB08] M. Pohl and P. Birke, *Interactive exploration of large dynamic networks*, Proc. 10th Int. Conf. on Visual Information Systems: Web-Based Visual Information Search and Management, VISUAL'08, Springer-Verlag, 2008, pp. 56–67. 74

[PCA02] H. C. Purchase, D. Carrington, and J.-A. Allder, *Empirical evaluation of aesthetics-based graph layout*, Empirical Software Engineering **7** (2002), 233–255. 76

[PCJ96] H. C. Purchase, R. F. Cohen, and M. James, *Validating graph drawing aesthetics*, Proc. ffSym. on Graph Drawing (GD'95), LNCS, vol. 1027, Springer-Verlag, 1996, pp. 435–446. 27, 74, 76

[PHG08] H. Purchase, E. Hoggan, and C. Görg, *How important is the "mental map"? - an empirical investigation of a dynamic graph layout algorithm*, Rev. Papers 15th Int. Sym. on Graph drawing (GD'07), LNCS, vol. 4875, 2008, pp. 184–195. 8, 75, 76

[PKL04] B. Pinaud, P. Kuntz, and R. Lehn, *Dynamic graph drawing with a hybridized genetic algorithm*, Automatic Computing in Design and Manufacture VI, Springer-Verlag, 2004, pp. 365–375. 27, 74, 77

[PL95] P. Peichen and C. L. Liu, *Area minimization for floorplans*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **14** (1995), no. 1, 123 –132. 28

[PMCC01] H. C. Purchase, M. McGill, L. Colpoys, and D. Carrington, *Graph drawing aesthetics and the comprehension of uml class diagrams: an empirical study*, Proc. Asia-Pacific Sym. on Information visualisation 2001, APVis'01, vol. 9, Australian Computer Society, Inc., 2001, pp. 129–137. 76

[PPP12] H. C. Purchase, C. Pilcher, and B. Plimmer, *Graph drawing aesthetics-created by users, not algorithms*, IEEE Trans. Vis. Comput. Graph. **18** (2012), no. 1, 81–92. 9, 27, 76

[PS08] H. Purchase and A. Samra, *Extremes are better: Investigating mental map preservation in dynamic graphs*, Diagrammatic Representation and Inference, LNCS, vol. 5223, Springer-Verlag, 2008, pp. 60–73. 8, 76

[PST97] A. Papakostas, J. M. Six, and I. G. Tollis, *Experimental and theoretical results in interactive orthogonal graph drawing*, Proc. 5th Sym. on Graph Drawing (GD'96), LNCS, vol. 1190, Springer-Verlag, 1997, pp. 371–386. 8, 74, 75

[PT90] F. N. Paulisch and W. F. Tichy, *Edge: an extendable graph editor*, Software: Practice and Experience **20** (1990), 63–88. 34, 70

[PT95] A. Papakostas and I. G. Tollis, *Improved algorithms and bounds for orthogonal drawings*, Proc. DIMACS Int. Workshop (GD'94), LNCS, vol. 894, Springer-Verlag, 1995, pp. 40–51. 22

[PT97] ———, *A pairing technique for area-efficient orthogonal drawings*, Proc. Sym. on Graph Drawing (GD'96), LNCS, vol. 1190, Springer-Verlag, 1997, pp. 355–370. 22

[PT00] J. Pach and G. Tóth, *Which crossing number is it anyway?*, Journal of Combinatorial Theory, Series B **80** (2000), 225–246. 44, 96

[Pur97] H. C. Purchase, *Which aesthetic has the greatest effect on human understanding?*, Proc. 5th Int. Sym. on Graph Drawing (GD'97), LNCS, vol. 1353, Springer-Verlag, 1997, pp. 248–261. 43, 74, 76, 96

[Pur98] ———, *The effects of graph layout*, Australasian Conference on Computer-Human Interaction (1998), 80. 74

[Pur00] ———, *Effective information visualisation: a study of graph drawing aesthetics and algorithms*, Interacting with Computers **13** (2000), no. 2, 477–506. 27, 76, 140

[Pur02] H. Purchase, *Metrics for graph drawing aesthetics*, J. Vis. Lang. Comput. **13** (2002), no. 5, 501–516. 9, 10, 27, 43, 74, 76, 81, 84, 91, 94, 96, 110, 139

[PVAE98] I. Park, M. Voss, B. Armstrong, and R. Eigenmann, *Parallel programming and performance evaluation with the ursa tool family*, Int. J. Parallel Program. **26** (1998), no. 5, 541–561. 6

[Rah99] M. S. Rahman, *Efficient algorithms for drawing planar graphs*, Ph.D. thesis, Tohoku University, Japan, 1999. 23

[RB06] G. Rote and I. Bárány, *Strictly convex drawings of planar graphs*, Documenta Mathematica **11** (2006), 369–391. 22

[Rei85] Gerhard Reinelt, *The linear ordering problem: Algorithms and applications*, Research & Exposition in Mathematics, 1985. 39

[RFF+08] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko, *Effectiveness of animation in trend visualization*, IEEE Trans. Vis. Comput. Graph. **14** (2008), no. 6, 1325 –1332. 76

[RNN98] M. S. Rahman, S.-I. Nakano, and T. Nishizeki, *Rectangular grid drawings of plane graphs*, Comp. Geo. Theory and Applications **10** (1998), 203–220. 23, 24

[RNN02] ———, *Rectangular drawings of plane graphs without designated corners*, Comp. Geo. Theory and Applications **21** (2002), 121–138. 23

[Rot12] G. Rote, *Realizing planar graphs as convex polytopes*, Proc. 19th Int. Sym. on Graph Drawing (GD 2011), LNCS, vol. 7034, Springer-Verlag, 2012, pp. 238–241. 22

[Rud93] R. Rudell, *Dynamic variable ordering for ordered binary decision diagrams*, IEEE/ACM Int. Conf. on Computer-Aided Design ICCAD'93, 1993, pp. 42–47. 50

[Saa01] Y. Saab, *A fast and effective algorithm for the feedback arc set problem*, Journal of Heuristics **7** (2001), 235–250. 39

[San95] G. Sander, *Graph layout through the vcg tool*, Proc. DIMACS Int. Workshop (GD'94), LNCS, vol. 894, Springer-Verlag, 1995, pp. 194–205. 67

[San96] ———, *A fast heuristic for hierarchical manhattan layout*, Proc. Sym. on Graph Drawing (GD'95), LNCS, vol. 1027, Springer-Verlag, 1996, pp. 447–458. 7, 60

[San99] ———, *Graph layout for applications in compiler construction*, Theoretical Computer Science **217** (1999), 175–214. 38, 60, 67

[SBOP08] J. Sobey, R. Biddle, P. C. Oorschot, and A. S. Patrick, *Exploring user reactions to new browser cues for extended validation certificates*, Proc. 13th European Sym. on Research in Computer Security: Computer Security (ESORICS'08), LNCS, Springer-Verlag, 2008, pp. 411–427. 140

[Sch90] W. Schnyder, *Embedding planar graphs on the grid*, Proc. 1st annual ACM-SIAM Sym. on Discrete algorithms, SODA'90, Society for Industrial and Applied Mathematics, 1990, pp. 138–148. 24

[Sco00] J. Scott, *Social network analysis: A handbook*, second. ed., Sage Publications, 2000. 3, 74

[SH07] M. Suderman and M. Hallett, *Tools for visually exploring biological networks*, Bioinformatics **23** (2007), no. 20, 2651–2659. 3

[Shn86] B. Shneiderman, *Designing the user interface: strategies for effective human-computer interaction*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986. 27

[SI10] J. Steele and N. Illinsky, *Beautiful visualization: Looking at data through the eyes of experts (theory in practice)*, 1 ed., O'Reilly Media, 2010. 114

[Smi09] M. Smith, *Peanut tainted product and producer networks, as seen in nodexl*, http://www.smrfoundation.org/2009/02/02/peanut-tainted-product-and-producer-networks-as-seen-in-nodexl/, 2009. 2

[SNM99] J. Stolfi, H. do Nascimento, and C. de Mendonça, *Heuristics and pedigrees for drawing directed graphs*, Journal of the Brazilian Computer Society **6** (1999), no. 1, 38 – 49. 140

[SP08] P. Saffrey and H. Purchase, *The "mental map" versus "static aesthetic" compromise in dynamic graphs: a user study*, Proc. 9th Conf. on Australasian user interface, AUIC'08, vol. 76, Australian Computer Society, Inc., 2008, pp. 85–93. 8, 9, 27, 75, 76

[SPS10] K. Soetaert, T. Petzoldt, and R. W. Setzer, *Solving differential equations in r*, The R Journal **2** (2010), no. 2, 5–15. 2

[SQ07] R. Shannon and A. J. Quigley, *Considerations in dynamic graph drawing: A survey*, http://rossshannon.com/publications/softcopies/rs2007-dynamic-graphs-survey.pdf, 2007. 74

[SR04] J. Shetty and M. Rey, *The enron email dataset database schema and brief statistical report 1*, http://www.isi.edu/~adibi/Enron/Enron_Dataset_Report.pdf, 2004. 2

[SSL+09] I. Surovtsova, N. Simus, T. Lorenz, A. König, S. Sahle, and U. Kummer, *Accessible methods for the dynamic time-scale decomposition of biochemical systems*, Bioinformatics **25** (2009), no. 21, 2816–2823. 2

[SSV95] F. Shahrokhi, L. Székely, and I. Vrt'o, *Crossing numbers of graphs, lower bound techniques and algorithms: A survey*, Proc. DIMACS Int. Workshop (GD'94), LNCS, vol. 894, Springer-Verlag, 1995, pp. 131–142. 44

[Ste51] S. K. Stein, *Covex maps*, Proc. American Mathematical Society **2** (1951), 464–466. 20

[Sto84] J. A. Storer, *On minimal node-cost planar embeddings*, Networks **14** (1984), 181–212. 22

[STT81] K. Sugiyama, S. Tagawa, and M. Toda, *Methods for visual understanding of hierarchical system structures*, IEEE Trans. Sys. Man & Cyber. **11** (1981), no. 2, 109–125. 7, 27, 33, 34, 47, 60, 70, 82

[Sug87] K. Sugiyama, *A cognitive approach for graph drawing*, Cybernetics and Systems **18** (1987), no. 6, 447–488. 7

[Sug02] ———, *Graph drawing and applications for software and knowledge engineers*, Series on software engineering and knowledge engineering, vol. 11, World Scientific, Singapore, 2002. 33

[SV97] F. Shahrokhi and I. Vrto, *Crossing numbers: Bounds and applications*, I. B'AR'ANY and K. BOROCZKY, Bolyai Society Mathematical Studies 6, Akademiai Kiado, 1997, pp. 179–206. 44, 96

[SWQN07] R. Shannon, G. Williamson, A. Quigley, and P. Nixon, *Visualising network communications to evaluate a data dissemination method for ubiquitous systems*, Proc. of Workshop on Ubiquitous Systems Evaluation in conjunction with Ubicomp'07, 2007. 8

[Tam87] R. Tamassia, *On embedding a graph in the grid with the minimum number of bends*, SIAM J. on Computing **16** (1987), 421–444. 22, 24

[Tam07] ———, *Handbook of graph drawing and visualization (discrete mathematics and its applications)*, Chapman & Hall/CRC, 2007. 3

[Tho84] C. Thomassen, *Plane representations of graphs*, Progress in graph theory (J.A. Bondy and U.S.R. Murty eds.), Academic Press, 1984, pp. 43–69. 23

[TMB02] B. Tversky, J. B. Morrison, and M. Betrancourt, *Animation: can it facilitate?*, Int. Journal of Human-Computer Studies **57** (2002), 247–262. 76

[TNB04] A. Tarassov, N. Nikolov, and J. Branke, *A heuristic for minimum-width graph layering with consideration of dummy nodes*, Experimental and Efficient Algorithms, LNCS, vol. 3059, Springer-Verlag, 2004, pp. 570–583. 43

[TPP09] R. Tamassia, B. Palazzi, and C. Papamanthou, *Graph drawing for security visualization*, Rev. Papers 16th Int. Sym. on Graph Drawing (GD'08), LNCS, vol. 5417, Springer-Verlag, 2009, pp. 2–13. 3

[Tri08] M. Trier, *Towards dynamic visualization for understanding evolution of digital communication networks*, Information Systems Research **19** (2008), no. 3, 335–350. 2

[TT86] R. Tamassia and I. G. Tollis, *Algorithms for visibility representations of planar graphs*, Proc. 3rd Annual Sym. on Theoretical Aspects of Computer Science (STACS'86), LNCS, vol. 210, Springer-Verlag, 1986, pp. 130–141. 24

[TTV91] R. Tamassia, I. G. Tollis, and J. S. Vitter, *Lower bounds for planar orthogonal drawings of graphs.*, Inf. Process. Lett. (1991), 35–40. 22

[Tut60] W. Tutte, *Convex representations of graphs*, Proc. London Mathematical Society **10** (1960), 304–320. 22

[Tut63] W. T. Tutte, *How to draw a graph*, Proc. London Mathematical Society **13** (1963), 743–767. 5

[UBSE98] J. Utech, J. Branke, H. Schmeck, and P. Eades, *An evolutionary algorithm for drawing directed graphs*, Proc. Int. Conf. on Imaging Science, Systems, and Technology 1998, CISST 1998, 1998, pp. 154–160. 57, 67, 140

[Urb12] Urbanrail.net, http://www.urbanrail.net/af/cairo/cairo.htm, 2012. 2

[VML96] V. Valls, R. Martí, and P. Lino, *A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs*, European Journal of Operational Research **90** (1996), no. 2, 303 – 319. 43

[Wag36] K. Wagner, *Bemerkungen zum vierfarbenproblem*, Jahresbericht der Deutsche Mathematiker Vereinigung **46** (1936), 26–32. 20

[War76] J. N. Warfield, *Societal systems: planning, policy, and complexity*, Wiley series on systems engineering and analysis, Wiley, 1976. 5, 6

[War77] ———, *Crossing theory and hierarchy mapping*, IEEE Transactions on Systems, Man, and Cybernetics **7** (1977), no. 7, 505–523. 7, 34, 43, 44

[WB04] C. Ware and R. Bobrow, *Motion to support rapid interactive queries on node–link diagrams*, ACM Trans. Appl. Percept. **1** (2004), 3–18. 74, 76

[Wid83] P. Widmayer, *Computational complexity in computer graphics and vlsi layout*, Ph.D. thesis, Department of Economics and Business Engineering, Karlsruhe University, 1983. 25

[Wol07] A. Wolff, *Drawing subway maps: A survey.*, Informatik - Forschung und Entwicklung **22** (2007), no. 1, 23–44. 3

[WS98] D. J. Watts and S. H. Strogatz, *Collective dynamics of "small-world" networks*, Nature **393** (1998), no. 6684, 440–442. 2

[YS99] A. Yamaguchi and A. Sugimoto, *An approximation algorithm for the two-layered graph drawing problem*, Proc. 5th annual Int. Conf. on Computing and combinatorics (COCOON'99), LNCS, Springer-Verlag, 1999, pp. 81–91. 5

[YT00] A. Yamaguchi and H. Toh, *Visualization of genetic neyworks: Edge crossing minimization of a graph drawing with vertex pairs*, Genome Informatics **11** (2000), 245–246. 43, 51

[YT01] ———, *Two-layered genetic neyworks drawings with minimimum edge crossings*, Genome Informatics **12** (2001), 456–457. 43, 51

[YYM12] C.-H. Yang, T.-H. Yu, and D. Markovic, *Power and area minimization of reconfigurable fft processors: A 3gpp-lte example*, IEEE Journal of Solid-State Circuits **47** (2012), no. 3, 757–768. 28

[ZKS11] L. Zaman, A. Kalra, and W. Stuerzlinger, *The effect of animation, dual view, difference layers, and relative re-layout in hierarchical diagram differencing*, Proc. of Graphics Interface GI'2011, Canadian Human-Computer Communications Society, 2011, pp. 183–190. 7, 76

[ZS08] H. Zhang and S. Sadasivam, *On planar polyline drawings*, Rev. Papers 15th Int. Sym. on Graph drawing (GD'07), LNCS, vol. 4875, 2008, pp. 213–218. 21

Zusammenfassung

Das *Graphzeichnen* (eng. Graph Drawing) befasst sich mit der geometrischen Darstellung von Graphen und wird von denjenigen Anwendungsbereichen verwendet, denen wichtig ist, die strukturellen Informationen als Graphen darzustellen. Der Großteil der Forschung im Bereich des Graphzeichens beschäftigt sich mit effizienten Algorithmen für die Arbeit mit statischen Graphen, die während der Ausführung eines Algorithmus die Struktur des Graphen nicht ändern. Bei dynamischen Graphen ändert sich die Struktur oder die Zeichnung des Graphen während der Ausführung eines Algorithmus durch Aktionen wie Hinzufügen, Entfernen oder Bewegen von Kanten und Knoten. Viele Graphenzeichner-Szenarien sind dynamisch, was eine ständige Aktualisierung des Graphen und schließlich dessen Neuzeichnung erfordert, nach dem eine gewisse Zahl von Änderungen ausgeführt sind.

Das Hauptziel im dynamischen Graphenzeichen ist die Bewahrung der kognitiven Landkarte ("mental map") des Benutzers und die Minimierung des Wiedererkennungsaufwands bei sich wiederholt ändernden Graphzeichnungen. Der empfohlene Ansatz zur Bewahrung der kognitiven Landkarte eines Benutzers ist, die Änderung zwischen der neuen und aktuellen Zeichnung zu minimieren. Dies erfordert adäquate Grundlagen für die Berechnung der Ähnlichkeit zwischen zwei Zeichnungen. Derartige Ähnlichkeitsmaße sind insgesamt als Differenzmetriken bekannt. Einige Differenzmetriken wie der Euklidische Abstand, relative Abstand, Kantenorthogonalität, Kantenbreite, Kantenanordnung werden hier üblicherweise hier betrachtet.

Diese Dissertation beschäftigt sich mit der Gestaltung und Validierung von Differenzmetriken beim Zeichnen von dynamisch veränderlichen hierarchischen Graphen. Die vorhandenen Ansätze in dieser Richtung fokussieren sich darauf, wie eine Aktion auf einem dynamischen hierarchischen Graph auszuführen ist und auf die Messung der Unterschiede zwischen Zeichnungen von dynamischen hierarchischen Graphen mit Hilfe von verschiedenen Metriken. Wir führen ein neues allgemeines Framework ein, in dem Differenzmetriken für dynamisch veränderliche hierarchische Graphen formuliert werden. Die allgemeinen Formen von Differenzmetriken werden entsprechend der topologischen und geometrischen Charakteristik von hierarchischen Graphen formuliert und auf die existierenden Metriken angewendet, die beim nicht-hierarchischen Graphenzeichen verwendet werden. Die vorgeschlagene Formulierung kann auf jeden hierarchischen Graphen von beliebiger Größe angewendet und auf verschiedenen Typen von Graphen erweitert werden.

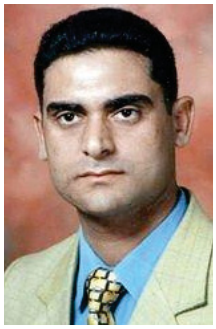
Messungen zur Ähnlichkeit zwischen zwei verschiedenen Graphen können allerdings nicht ausschließlich auf Basis formaler Metriken durchgeführt werden. Zur Untersuchung des ästhetischen Empfindens bzw. der wahrgenommenen Änderungen der "mental map" sind direkte Rückmeldungen von menschlichen Nutzern erforderlich. Zu diesem Zweck wurde eine

experimentale Benutzerstudie durchgeführt, um die eingeführten Differenzmetriken basierend auf die Benutzerbewertungen zu überprüfen.

Als weiteres Problem des Graphzeichnens wurde die Minimierung von Kreuzungen zwischen den Kanten beim Zeichen von hierarchischen Graphen. Wir führen einen neuen Algorithmus mit dem Namen „efficient barycenter“ ein, der bessere Ergebnisse sowohl bei der Anzahl der Kreuzungen als in der Literatur bekannte Algorithmen.

Ein weiterer Beitrag dieser Arbeit ist eine neue Methode zur Generierung zufälliger hierarchischer Graphen als Grundlage für die experimentellen Untersuchungen. Sie kontrolliert alle Parameter eines hierarchischen Graphen, wie die Anzahl der Ebenen, die Anzahl der Knoten in jeder Ebene, die minimale Anzahl der ausgehenden Kanten eines Knoten, die Kantendichte und das Verhältnis der langen Kanten in einem Graph.

Curriculum Vitae



Name	ISMAEEL, Alaa Aly Khalaf
Nationality	EGYPTIAN
Address	• Kronenstr. 1, 76133 Karlsruhe, Germany • Computer Science Dept., Minia University, 61519 El-Minia, Egypt
Phone	+49(0)176 6259 6659, +49(0)721 20 888 67, +20(0)100 760 2487
Email	• alaa.ismaeel@kit.edu • alaa.ismaeel@science.miniauniv.edu.eg
Date of Birth	14.12.1975
Place of Birth	El-Minia, Egypt
Marital Status	Married (with 3 children)
Military Status	Completely Exempted
Education	• **1981-1987** Al-Horeya Primary School, Matay, El-Minia, Egypt • **1987-1990** Al-Tahrir Prip School, Matay, El-Minia, Egypt • **1990-1993** Matay Secondary School, Matay, El-Minia, Egypt • **1994-1998** Minia University, El-Minia, Egypt
Jobs	• **Dec. 1999** Demonstrator (Computer Science), Computer Science Dept., Faculty of Science, Minia University, El-Minia, Egypt • **Jun. 2004** Assistant Lecturer (Computer Science), Computer Science Dept., Faculty of Science, Minia University, El-Minia, Egypt • **Mar. 2008** Research Associate, Institute AIFB, Karlsruhe Institute of Technology (KIT), Germany

Karlsruhe, Mai 2012


الحمد لله رب العالمين