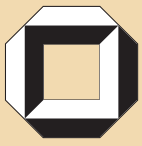
Markus Franke

An Update Algorithm for Restricted Random Walk Clusters



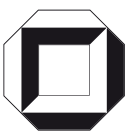
universitätsverlag karlsruhe

Markus Franke

An Update Algorithm for Restricted Random Walk Clusters

An Update Algorithm for Restricted Random Walk Clusters

by
Markus Franke



universitätsverlag karlsruhe

Dissertation, Universität Karlsruhe (TH)
Fakultät für Wirtschaftswissenschaften, 2007
Tag der mündlichen Prüfung: 28.02.2007
Referenten: Prof. Dr. Andreas Geyer-Schulz, Prof. Dr. Karl-Heinz Waldmann

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe
www.uvka.de

Universitätsverlag Karlsruhe 2007
Print on Demand

ISBN: 978-3-86644-183-5

Contents

Chapter 1

Motivation

Being able to quickly evaluate newly arriving data or information is a crucial capability in today's economy. But this evaluation may be difficult, especially if the new data cannot be interpreted in an isolated way but rather is added to a data base where it may lead to the update of preexisting records or the creation of new ones. An excellent example of such a situation is a purchase data base maintained by a retailer: In regular intervals, new purchases are integrated into the data base which, in turn, may influence the marketing and product portfolio decisions made by the retailer.

In a static scenario, cluster algorithms are used among other methods to reduce the complexity of such a data set and to identify for instance groups of customers with similar purchase histories. For large data sets, the execution of a cluster algorithm may take considerable time and computing resources [Vie97]. These time and resource requirements may be acceptable for static information, when the data set only needs to be clustered once. But what if the data is updated frequently? In most cases, a reclustering of the whole data set will be both impractical as well as not economical, especially for small updates. Since in this case large parts of the data set are not changed by the update, valuable time is lost while the cluster algorithm recomputes clusters that were not affected by the update.

This problem constitutes the motivation for this thesis. In earlier contributions [Fra03, FGS04, FGS05, FT05], the restricted random walk (RRW) cluster algorithm developed by Schöll and Schöll-Paschinger [SP02] was evaluated in the context of a large data set of library purchase histories and found to work well for that data set in static scenarios, both in terms of the quality of the clusters and of the computation time required. The challenge for this work is the integration of new data into the cluster structure with minimal computational effort. As a further condition, the cluster quality should remain the same whether the new data is integrated using the update algorithm or by reclustering the complete data set.

New data in this context can mean one of three different things:

1. New objects may enter the data set, along with information about their similarity or distance to other, existing objects.

2. An object may change its similarity or distance to other objects, either by physically moving to another location or by changing its characteristics that determine its similarity to other objects.

3. An object may be removed from the set.

We will see that, contrary to other methods reviewed in chapter 2, the update procedure proposed here is able to handle all three cases within reasonable computational time while maintaining the cluster quality.

This thesis is structured as follows: In the remainder of the first chapter, terms used in this work are defined. Chapter 2 contains an overview of the current state of research in the area of stochastic processes and cluster algorithms. In chapter 3 the RRW clustering method is introduced as a Markov chain on a similarity or distance graph. In addition, the chapter contains applications for which the algorithm has been successfully used as well as some considerations about the algorithm's complexity and the asymptotic behavior of the walk process.

The core of this thesis, the update algorithm, is developed in chapter 4 together with a proof of its correctness and a comparison with the algorithm classes introduced in chapter 2. Chapter 5 concludes the thesis with an outlook.

1.1 Definitions

A very fitting definition for cluster analysis has been given by Kaufman and Rousseeuw [KR90]: "Cluster analysis is the art of finding groups in data." Although this view on cluster analysis may seem informal, the informality is justified. There exists – to the practitioner's regret – no "best" cluster algorithm that copes with all applications on all data sets equally well. Rather, it depends on the characteristics of the data set as well as on the requirements of the concrete application at hand what a good cluster is and, consequently, a plethora of criteria [HK01, HBV02b, HBV02a, PL02, War63, Wat81] has been proposed to measure the quality of different cluster algorithms.

As we are not able to give a global exact definition, let us at least consider a sort of least common denominator. A cluster, according to the Oxford Advanced Learner's Dictionary [Cow89, p. 215], is a

> 1 number of things of the same kind growing closely together: *a cluster of berries, flowers, curls ∘ ivy growing in thick clusters.* 2 number of people, animals or things grouped closely together: *a cluster of*

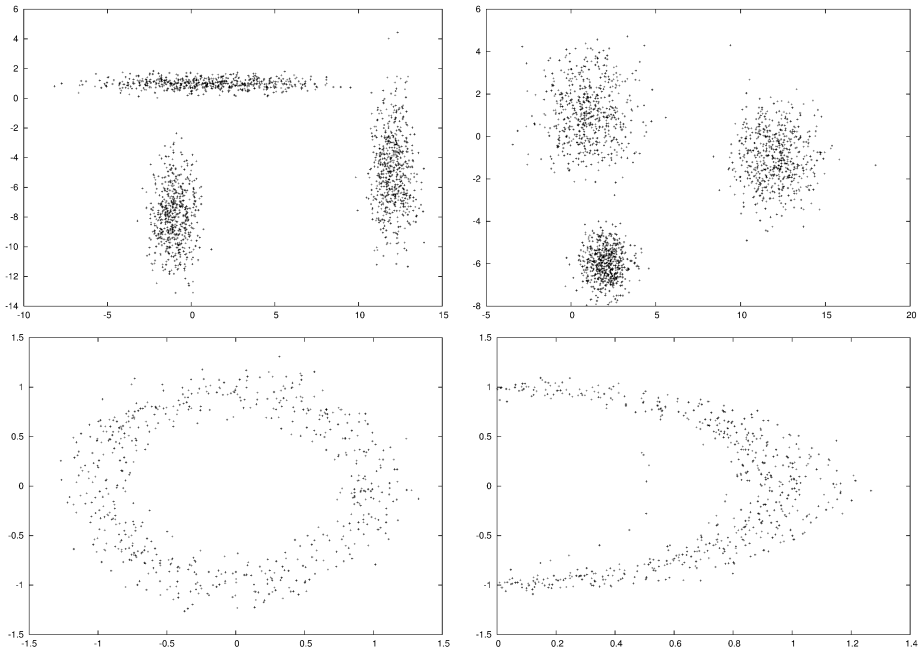


Figure 1.1: Cluster shapes: (a) elongated, (b) compact, (c) ring, (d) sickle

> *houses, spectators, bees, islands, diamonds, stars* ∘ *a consonant cluster,* eg *str* in *strong.*

In the context of this work, the following definition (modified from [Fra03]) is used:

Definition 1.1.1 (cluster) *A cluster is a set of objects that are either (a) similar to each other or (b) close – in the sense of some metric – to each other. Especially objects in one cluster should be more similar to each other (intra-cluster homogeneity) than to objects in any other cluster (inter-cluster heterogeneity).*

In this context, the intra-cluster criterion can be based for example on diameter, radius (cf. page 25), variance (cf. page 53), or the sum of squared errors, i.e. the variance multiplied by the number of elements in the cluster. The actual quality of a clustering can consequently be given as a function of this criterion.

However, the definition does not make any assumptions concerning the actual shape of a cluster. In the literature (e.g. [KR90]) the shapes depicted in Fig. 1.1 are often considered as standard cases. Depending on the shape of the clusters, the performance of different cluster algorithms – in terms of the chosen quality criterion – may vary considerably. For instance, there are algorithms like k-means clustering that are especially fit to detect spherical objects like the ones in case

(b), and that thus cannot cope well with elongated clusters like case (a) in Fig. 1.1. Others, like single linkage clustering, have the problem of bridging: If, between two clusters, there exists a "bridge" of outliers, the two clusters may be connected via this weak link, though it does not accurately reflect the natural grouping.

In addition to a cluster, a clustering is defined as follows:

Definition 1.1.2 (clustering) *A clustering is the result of the execution of a cluster algorithm on an object set X. It describes the assignment of objects to clusters. X, the set of objects to be clustered, is said to be covered by the clustering C consisting of clusters C_i iff*

$$X = \bigcup_{C_i \in C} C_i \tag{1.1}$$

Furthermore, a clustering created by a cluster algorithm can either be disjunctive or not and it can be hierarchical or partitional:

Definition 1.1.3 (disjunctive clusters) *We call a set of clusters disjunctive if each object belongs to exactly one cluster, i.e. if the clusters do not overlap. In this case, the following property holds:*

$$\forall C_i, C_j \in C : \; C_i \cap C_j = \emptyset \tag{1.2}$$

If a cluster algorithm produces disjunctive clusters for every object set, it is also called disjunctive.

Definition 1.1.4 (partitional clustering) *A partitional cluster algorithm generates a clustering containing k disjunctive clusters; the resulting clustering is also called a k-partitioning. k is usually fixed in advance.*

Definition 1.1.5 (hierarchical clustering) *A hierarchical cluster algorithm produces a hierarchy of clusters that can be represented by a dendrogram as given by definition 1.1.7. The hierarchy includes clusters at different aggregation levels, with an all-encompassing cluster, often called root cluster at the top. At the bottom of the dendrogram, each object is contained in its own cluster. Hierarchical cluster algorithms may work bottom-up (agglomeratively) by successively merging clusters or top-down (divisively) by iteratively splitting up clusters.*

It is clear that the clustering at each level of a hierarchical clustering is a partition of the data set if the clusters are disjunctive. For agglomerative hierarchical cluster algorithms, the building blocks to agglomerate are so-called singleton clusters:

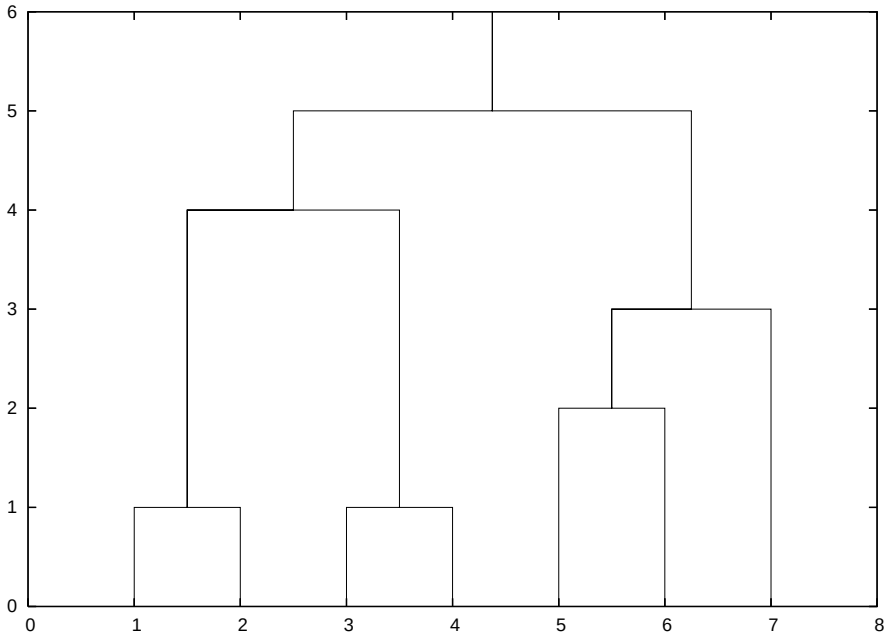


Figure 1.2: A dendrogram

Definition 1.1.6 (singleton cluster) *A singleton or singleton cluster is a cluster consisting of exactly one object.*

Definition 1.1.7 (dendrogram) *A dendrogram is the graphical tree representation of the hierarchy of clusters produced by a disjunctive hierarchical cluster algorithm at different levels. Normally, the root cluster representing the entire object set is located at its top, the singleton clusters at the bottom, as can be seen in Fig. 1.2. The horizontal lines represent the joins when ascending in the tree – respectively the splits when descending – that occur when changing between levels of the clustering.*

Up to now, we have been talking about similarity (or distance) as an abstract concept. In order to develop a formal definition, let us introduce two different concepts for the representation of the objects and their relation: The vector and the graph model.

The vector model constitutes the most natural case of some metric space, for instance a vector space with a given norm. The objects are represented as vectors, with each vector element or dimension corresponding to a specific feature of the object. These features can either be binary, nominal (qualitative) or quantitative. An object set in the vector model is given by a two-mode matrix; a two-mode matrix represents the objects as rows and the features as columns. Thus, the objects

may also be seen as row vectors. It is then possible to define a distance measure on this representation.

Definition 1.1.8 (distance) *A distance measure on an object set X is a relation $d : X \times X \rightarrow \mathbb{R}$ with the properties (1.3) to (1.6) that assigns a distance $d(u, v)$ to each pair u, v of objects in X.*

$$d(u, v) \geq 0 \tag{1.3}$$

$$d(u, u) = 0 \tag{1.4}$$

$$d(u, v) = d(v, u) \tag{1.5}$$

$$d(u, v) \leq d(u, w) + d(w, v) \tag{1.6}$$

Eq. (1.3) asserts that the distance is non-negative. The distance of an object to itself is zero (Eq. (1.4)), as well as to all other objects occupying the same point in space. Furthermore, the distance is symmetric (Eq. (1.5)) and complies to the triangle inequality (Eq. (1.6)).

Examples for distance functions are the Euclidean distance $d(u, v) = \sqrt{\sum_{i=1}^{k}(|u_i - v_i|^2)}$ or the city-block metric $d(u, v) = \sum_{i=1}^{k} |u_i - v_i|$, both given here for k dimensions. They belong to the family of L^q or Minkowski distances whose members have the form

$$d_q(u, v) = \sqrt[q]{\sum_{i=1}^{k}(|u_i - v_i|^q)} \tag{1.7}$$

for k dimensions. For linearly dependent features, the Mahalanobis distance [Mah36]

$$d(u, v) = (u - v)\Sigma^{-1}(u - v)^T \tag{1.8}$$

should be used where Σ^{-1} is the inverse of the covariance matrix and $(u - v)^T$ is the transpose of $(u - v)$. It has the advantage of being invariant to scale transformations and translations of the raw data.

The second model for object representation is the graph model where objects are not represented by their absolute position in a metric space, but rather using a one-mode matrix that contains the strength of their relations to the other objects, either as dissimilarity or as similarity.

Definition 1.1.9 (dissimilarity) *A dissimilarity measure on an object set X is a relation with the properties (1.3) to (1.5) that assigns a dissimilarity to each pair of objects in X. It does not necessarily comply to (1.6), i.e. the triangle inequality may be violated.*

Definition 1.1.10 (similarity) *A similarity measure on an object set X is a relation $s : X \times X \rightarrow \mathbb{R}$ with the properties (1.9) to (1.11) that assigns a similarity to each pair of objects in X.*

$$0 \leq s(u, v) \leq 1 \tag{1.9}$$

$$s(u, u) = 1 \tag{1.10}$$

$$s(u, v) = s(v, u) \tag{1.11}$$

Similarities are usually – but not necessarily – normed to the interval $[0, 1]$ (Eq. 1.9). The self-similarity is set to one respectively the highest possible value (Eq. 1.10) – normally, cluster algorithms do not use this value. The similarity relation is symmetric (Eq. 1.11). The case of directed graphs with asymmetric similarities is not considered in this thesis.

In the context of information retrieval, the cosine measure is often used to obtain similarities between objects represented as vectors in a metric space, each vector containing in its components the features of the object it represents.

Definition 1.1.11 (cosine similarity) *The cosine similarity between two objects u and v represented as vectors in a k-dimensional vector space is given as the cosine of the angle between their vectors*

$$cos(\Theta(u, v)) = \frac{|u \cdot v|}{||u|| \cdot ||v||} \tag{1.12}$$

Here, $|u \cdot v| = \sum_{i=1}^{k} u_i v_i$ is given as the scalar product of u and v, and for the length or norm $||u|| = \sqrt{\sum_{i=1}^{k} u_i^2}$ of u the Euclidean vector norm is used.

Finally, another useful group of tools for modelling input data sets should be introduced: graphs [Die05, Jun99].

Definition 1.1.12 (graph) *A graph $G = (V, E, \omega)$ is defined by the set of its vertices V, the set $E \subseteq V \times V$ of edges and the matrix $\omega = (\omega_{ij})_{|V| \times |V|}$ containing the edge weights between the pairs of nodes i and j. Graphs can be undirected or directed: In an undirected graph, if node A has an edge leading to node B, B is considered to also be connected to A. This is not necessarily true for directed graphs.*

If the edge weights represent the pairwise similarities between the respective nodes, the graph is also called a similarity graph. Graphs may contain cycles in the form $(i_1, i_2, \ldots, i_k)$ with $i_k = i_1$; $i_l \in V \; \forall 1 \leq l \leq k$; $i_l \neq i_m \forall 1 \leq l, m < k \wedge l \neq m$, and $k > 2$, i.e. sequences that end at their start node and contain no repeated nodes with the exception of the start/end node. A special case of a graph is a tree:

Table 1.1: Raw data for the example graph

	A	B	C	D	E	F	G
A	–	5	6	1	–	–	–
B	5	-	4	–	–	2	2
C	6	4	–	–	–	–	7
D	1	–	–	–	–	1	–
E	–	–	–	–	–	–	6
F	–	2	–	1	–	–	–
G	–	2	7	–	6	–	–

Definition 1.1.13 *A tree is a cycle-free graph (undirected or directed).*

The reason why graphs are often used in the context of cluster analysis is that they offer an intuitive approach to the exploration of a given data set and a standardized format as well as a tool box for its analysis. When considering a graph in its graphical representation like the one in Fig. 1.3, it is often possible to gain a first idea of what "natural" clusters should look like by identifying densely populated regions (where a metric exists that allows such statements) or regions with high similarities. Furthermore, the graph representation is optimal for data sets where only a similarity or dissimilarity measure is available; the objects in such a data set can be mapped to the vertices, and the (dis)similarity measure is reflected by the edge weights. For instance, the data set used for the construction

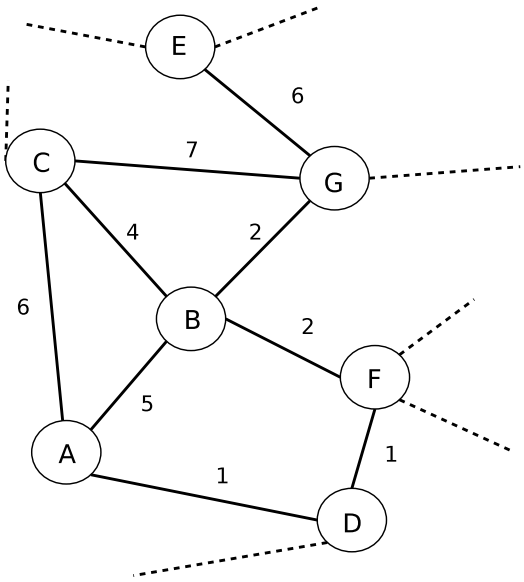


Figure 1.3: Section of an example similarity graph

of the example graph in Fig. 1.3 is given in Tab. 1.1, where the entries in the cells represent the pairwise similarities between two nodes or objects. If there is no entry, the similarity is not defined explicitly and can be assumed to be minimal – usually zero; as a consequence, the respective nodes are not connected by an edge.

In our example, the group ABC can be interpreted as a cluster since all its members are connected with relatively high similarities. Equally, there is an important relation between nodes C, G, and E due to the high similarities between them. For instance, CG and GE might form a cluster. If we assume transitivity of relations – for instance in social networks, where the graph either is incomplete due to observation errors or may develop new edges over time – the three nodes CGE might also form a cluster. In that case, the high similarity between C and G on the one hand and G and E on the other allow to deduce a "considerable" similarity between C and E; as a consequence, the three nodes form a cluster. We will see later on, in the third chapter, that the RRW algorithm also finds such transitive relations.

Chapter 2

Related Work

This chapter will provide an overview of the literature that is relevant in the context of this thesis.

An important concept for the restricted random walk cluster algorithm is that of a stochastic process. Therefore, section 2.1 contains an introduction to stochastic processes, with an emphasis on Markov chains, a specific and well-known class of stochastic processes. The restricted random walk as defined by Schöll and Schöll-Paschinger [SSP03] is introduced, and its definition is delineated from other restricted random walk concepts found in the literature.

In addition to Markov chains, random graphs are another interesting construct with close ties to the subject of random walks. Section 2.2 gives a short introduction into the subject.

The remainder of the chapter will review cluster algorithms with an emphasis on two areas: "dynamic" clustering and randomized algorithms. In the context of this thesis, the term dynamic clustering will encompass those cluster algorithms that are fit to cope with data that is – in one sense or another – dynamic. More precisely, this includes data sets whose object set may vary in size, as well as data sets with changing distances, dissimilarities or similarities.

As an introduction into the subject of clustering, section 2.3 starts with a short survey of cluster algorithms, before section 2.4 presents the different approaches that can be summarized under the label of dynamic clustering. These approaches are divided into three subclasses. The most established algorithm class is the one in section 2.4.1 that deals with growing data sets and data streams. The description of truly dynamic algorithms that are able to work on data sets in which similarities or distances can change follows in section 2.4.2. A special subclass of such dynamic problems can be found in section 2.4.3: mobile scenarios, where clustering based on visibility criteria is used to e.g. optimize routing in wireless networks.

The following two sections are dedicated to the stochastic aspects of clustering: Section 2.5, after giving a short introduction to random algorithms, presents

a number of probabilistic algorithms for clustering, whereas section 2.6 discusses approaches that are deterministic, but based on ideas from random walk theory.

It should be noted here that this thesis uses the original variable names proposed by the authors of the literature reviewed. While this facilitates a later comprehension of the original work, it also implies that the scope of the variable definitions in the sections 2.3 to 2.5 is always limited to the respective section or paragraph.

2.1 Stochastic Processes

According to Ferschl [Fer70], a stochastic process in general is a mathematical model for randomized experiments over a probability space that consists of a sequence of observations or states. This sequence is represented by a succession $\{X_t,\, t \in T\}$ of states that are visited by the process at times t. T is called the parameter space of the process and represents the time. In the case of the discrete state space $T = \mathbb{N}_0$, transitions between the states may occur at given intervals; the process is then called discrete. In other cases like $T = \mathbb{R}$, the process may change its states at any point in time; it is called continuous. The set containing the possible states, i.e. the possible values of the X_t, is denoted by S and called the state space. It may either be finite or infinite. We will deal here only with discrete processes with a finite state space.

A concrete realization of such a process may be written as $X = (x_0, x_1, x_2, \ldots)$, where each x_k can be interpreted as a realization of a random variable X_k. The process itself is defined by the parameters S, x_0, and P, where S is the aforementioned set of states, x_0 is the initial state of the process, and P denotes the transition probabilities, i.e. the probabilities of entering a certain state x_k after the process has been in the states $x_0, \ldots, x_{k-1}$.

2.1.1 Random Walks

The term random walk was first introduced in a contribution to the Nature magazine in 1905 by Pearson titled “The problem of the random walk” [Pea05]:

> Can any of your readers refer me to a work wherein I should find a solution of the following problem[...]. A man starts from a point O and walks l yards in a straight line; he then turns through any angle whatever and walks another l yards in a second straight line. He repeats this process n times. I require the probability that after these n stretches he is at a distance between r and $r + \delta r$ from his starting point, O.

> The problem is one of considerable interest [...]

Given that last sentence it is not surprising that Pearson's question was not the first mention of this problem and that it was answered by Lord Rayleigh in the same magazine shortly after [LR05]. Rayleigh cited the solution he had found in 1880 for the analysis of a large number of harmonic vibrations [LR80]. Over the last century, there has indeed been a considerable interest in the problem of the random walk. Bachélier [Bac00] had considered a similar question in his analysis of speculation, investigating for instance the probability that the price of a bond reaches a certain limit at a given time or the expectation value for the first time the price exceeds a predetermined level. Instances from the same problem class were also discussed by Markov [Mar12] who, among others, presented the solution to the ruin problem in a two-player game that will serve as example in the following section.

There are different formulations of random walks. The basic formulation as given by Cox and Miller [CM65] is that of a particle (the walker) in a multidimensional space. If X_0 is the (random) vector denoting its initial position and the walker's displacement in the l-th step is denoted by r_l, its position after n steps is $X_n = X_0 + \sum_{k=1}^{n} r_k$. The r_k are independently (and in many cases identically) distributed random variables with density $p_k(r_k)$. If the r_k – and with them, the X_n – are defined on a discrete set, the resulting process is called a lattice walk as introduced by Polya [Pol21].

Furthermore, if the process is defined in continuous time and the time interval between two consecutive steps approaches zero, the resulting process converges to a Brownian motion [BN70].

Random walks have received considerable interest as predicted by Pearson because they can be used to model, analyze, and predict the behavior of many systems where random events play a role. Examples include insurance problems with ruin models or games [Spi01] as will be exemplified in section 2.1.4.

2.1.2 The Restricted Random Walk Used for Clustering

In order to give a first idea of the random process underlying the RRW clustering algorithm, this section will introduce the restricted random walk process as proposed by Schöll and Schöll-Paschinger [SP02, Sch02, SSP03]. In chapter 3, these ideas are presented in much deeper detail, and the use of restricted random walks for clustering purposes is described.

Consider a walk process in a metric space that is populated with objects i belonging to some finite set X. Let the process start at some object i_0. It picks a successor object i_1 from a uniform distribution over all other objects in X. For the second step, a restriction is introduced: The next object has to be closer to i_1

than i_0 is. If $d(i, j)$ is the distance relation associated with the metric space, this means that the restriction

$$d(i_k, j) > d(i_k, i_{k-1}) \tag{2.1}$$

has to be fulfilled for all $k \geq 1$. If there is no object that satisfies the restriction in Eq. (2.1), the walk ends. It follows from the construction that such a walk must be finite on a finite object set.

This formulation of the restricted random walk is the one that is used in this thesis. As can be expected, the term restricted random walk has not been used exclusively by Schöll and Schöll-Paschinger for their algorithm. Literature mentions other types of restricted random walks. In order to provide a delineation to these concepts, the next section includes other definitions for restricted random walks; these will not be used again in this thesis.

2.1.3 Other Concepts of "Restricted Random Walks"

Many publications, for instance [BN70, WK99, Yan05], use the label restricted random walks for self-avoiding walks on graphs that are random walks with a different kind of restriction: Instead of requiring an increasing similarity between successive nodes as does the RRW method discussed here, a restricted random walk in the sense of these publications is a walk that may not return to a node it has already visited. This restriction applies either for the last k steps or for the whole walk. Formally, a restricted random walk in the sense of this definition has the form $(i_0, \ldots, i_n | i_l \neq i_h \forall l \neq h \wedge 0 \leq l, h \leq n)$ (strict formulation) respectively $(i_0, \ldots, i_n | i_l \neq i_h \forall l \neq h \wedge |l - h| \leq k \wedge 0 \leq l, h \leq n)$ (last k nodes forbidden). The choice of a successor among the admissible nodes usually follows a uniform distribution. The advantage of this formulation is that the process has the stochastic exploration capabilities of a random walk, but is forced to visit and explore unknown nodes instead of reiterating over already known ones. As a consequence, the probability is high that a restricted random walk in this sense explores larger portions of a graph than its non-restricted counterpart. On the other hand, if the threshold k is not well chosen, the random walk might become trapped in an isolated region of the graph that is only connected to the rest by a node already visited by the process. The use of such restricted random walks for the exploration of graphs has for instance been discussed by Yang [Yan05].

Engelberg [Eng65] has scrutinized specific walks of length $a + b$. These walks are designed to move along the x-axis; their movement along the y-axis is a random choice between -1 and 1. The walker – called particle – starts at $(0, 0)$, and its vertical movement in the k-th step is given by a random variable $\xi_k \in \{-1, 1\}$. The position after the k-th step is thus $(k, \sum_{i=1}^{k} \xi_i)$. The restriction that is imposed

on the walks in this case is that the number of positive movements is exactly a, and that of negative ones is exactly b. For these, Engelberg developed the probability distributions for the number of zeros, i.e. visits to the x-axis and the number of crossings of the walk, i.e. the number of times the process actually crosses the x-axis.

Gray [Gra68] has generalized these findings to the distribution of the number of visits and crossings at an arbitrary y-level.

2.1.4 Markov Chains

Markov chains are generally considered well-behaved and well-investigated stochastic processes. As will be shown in section 3.1.1, restricted random walks can be modeled as Markov chains in discrete time, thereby allowing access to the complete toolbox offered by Markov theory. A comprehensive overview of these tools can be found e.g. in the books by Ferschl [Fer70] or, in a more general context, Karr [Kar93].

A stochastic process $X = (x_0, x_1, x_2, \ldots)$ is called a Markov chain if it is compatible with the Markov condition that the choice of the successor state only depends on the current state, i.e.

$$P(X_k = j | X_0 = x_0, \ldots, X_{k-1} = x_{k-1}) = P(X_k = j | X_{k-1} = x_{k-1}) \quad \forall k \in \mathbb{N}, j \in S \tag{2.2}$$

A Markov chain is called homogeneous if $P(X_k = j | X_{k-1} = x_{k-1})$ is independent of the parameter k, in other words, if

$$P(X_k = j | X_{k-1} = x_{k-1}) = P(X_l = j | X_{l-1} = x_{l-1}) \quad \forall k, l \in \mathbb{N},\ j \in S \tag{2.3}$$

In that case, the transition probabilities are denoted as a stochastic matrix $P \in \mathbb{R}^{S \times S}$. For the rest of the thesis, it will be assumed that the Markov chains under consideration are homogeneous. This is justified since updates never occur during a walk, but always between executions of the algorithm; consequently, the Markov chain constructed in each execution of the algorithm is homogeneous.

A typical example of a Markov chain is a dice game: A person with an initial endowment $n_0 < X_0 < n_1$ throws a dice and obtains one monetary unit (MU) if the number on the dice is greater than 3 and loses one MU if it is less or equal 3. Each round's payoff is a variable $Y_k \in \{-1, 1\}$ and the state variable in this case is the person's wealth $X_k = X_0 + \sum_{l=0}^{k} Y_l = X_{k-1} + Y_k$. Obviously X_k only depends on the last state and the result of the dice throw.

There are several conceivable state spaces for this game: $\mathbb{Z}$, $\mathbb{N}_0$, or sets formed by an interval $[n_0, n_1]$ with $n_1 - n_0 \in \mathbb{N}$. In the first case, the player is granted unlimited credit and the game may go on infinitely. If, in contrast, credit is not

granted ($S = \mathbb{N}_0$), the game will end at some point with the ruin of the player when $X_k = 0$. Finally, if the state space is limited on both sides due to the player quitting when an endowment of n_1 is reached ($S = \{n_0, n_0 + 1, \ldots, n_1 - 1, n_1\}$), the transition probabilities are given by

$$P(X_k = j) = \begin{cases} \frac{1}{2} & \text{if } j \in \{x_{k-1} - 1, x_{k-1} + 1\} \text{ and } n_0 < x_{k-1} < n_1 \\ 1 & \text{if } x_{k-1} \in \{n_0, n_1\} \text{ and } j = x_{k-1} \\ 0 & \text{else} \end{cases} \tag{2.4}$$

In this formulation, the states n_0 and n_1 are absorbing states, i.e. once the process enters one of these states, it cannot leave it anymore. For the player in the example entering the state n_0 represents his ruin. Often, for instance for insurance companies, the probabilities connected to the ruin are of interest. Let J be a set of absorbing states. T_J denotes the first time the process enters one of the states contained in J; the probability that T_J is finite, i.e. in our case, that the ruin occurs in finite time when the player starts in state i can be calculated as the solution of the equation system

$$P_i(T_J < \infty) = \sum_{k \in J} p_{ik} + \sum_{k \notin J} p_{ik} P_k(T_J < \infty) \tag{2.5}$$

with $0 \leq P_i(T_J < \infty) \leq 1$. For the following computations, $n_0 = 0$ and $n_1 = 5$ shall be assumed.

The probability of the ruin of our player in finite time is computed by setting up the equation system above for $J = \{0\}$ that is given here with P_i as abbreviated notation of $P_i(T_J < \infty)$. The right hand side contains the probability of directly entering the absorbing state n_0. It follows from the definition of the process.

$$\begin{array}{rcrcrcrcrcl} P_0 & & & & & & & & & = & 1 \\ & & P_1 & - & \frac{1}{2}P_2 & & & & & = & \frac{1}{2} \\ - & \frac{1}{2}P_1 & + & P_2 & - & \frac{1}{2}P_3 & & & & = & 0 \\ & & - & \frac{1}{2}P_2 & + & P_3 & - & \frac{1}{2}P_4 & & = & 0 \\ & & & & - & \frac{1}{2}P_3 & + & P_4 & - \frac{1}{2}P_5 & = & 0 \\ & & & & & & & & P_5 & = & 0 \end{array}$$

Solving the equation system, we see that for instance the probability of ruin is 0.4 if the player starts with an initial endowment of $X_0 = 3$. The probabilities of reaching the state n_1 instead are calculated analogously.

Additionally, if the process reaches an absorbing state almost surely, i.e. with probability 1, in finite time, it might be interesting to compute the time that the process takes on average to terminate. If J is once again the set of absorbing states, the expected number of steps $E_i(T_J)$ the process takes from $X_0 = i$ before

entering an absorbing state is the solution of the equation system

$$E_i(T_J) = 1 + \sum_{k \notin J} p_{ik} E_k(T_J)$$

with $E_i(T_J) \geq 0 \; \forall i$.

If the state space of the game is not limited, the process is a random walk on a one-dimensional lattice where the state X_n is defined as $X_n = X_{n-1} + Y_n$ and

$$P(Y_n = l) = \begin{cases} \frac{1}{2} & \text{if } l \in \{-1, 1\} \\ 0 & \text{else} \end{cases} \tag{2.6}$$

Using the one-step transition probability matrix P, it is possible to calculate the probability distribution after a given number of steps. First, let the row vector $\pi_t \in [0,1]^{|S|}$ represent the probability distribution of the states at time t. The distribution for $t+1$ is obtained by multiplication

$$\pi_{t+1} = \pi_t P \tag{2.7}$$

and by recursion, we get

$$\pi_{t+l} = \pi_t P^l \tag{2.8}$$

A special case occurs if, starting from a time t_0, the distribution does no longer change from step to step:

Definition 2.1.1 (stationary distribution) *If $\pi_t = \pi_{t+1} = \pi$ for all $t \geq t_0$, the distribution π of the Markov chain is called stationary or "steady state".*

A Markov chain is bipartite if its state space S can be split into two subgroups S_1, S_2 such that $S_1 \cup S_2 = S$ and $p_{ij} = 0 \, \forall i \in S_1, j \in S_1, \vee \, i \in S_2, j \in S_2$, i.e. if there are no transitions between two members of the same subgroup. As Lovasz [Lov96] has shown, all non-bipartite Markov chains have a stationary distribution. Bipartite chains, on the other hand, cannot have a stationary distribution because with each step, the transition probabilities change radically, depending on which part of the state space the current state belongs to.

Another important notion is that of (ir)reducibility.

Definition 2.1.2 (irreducible Markov chain) *A Markov chain is said to be irreducible if each of its states can be reached from each other state, i.e. if no part of the state space can get separated from the rest during the running time of the process.*

2.2 Random Graphs

Closely related to random walks on graphs is the subject of random graphs that goes back to Erdös and Renyi [ER57, ER60]. In a random graph, the vertices are predetermined; their edges, on the contrary, are created by a stochastic process.

For instance, consider a party where guests walk randomly through the room and engage in dialogs for a limited time with the first person they meet and that currently is not having a conversation. Clearly, the nodes of the graphs are the persons participating in the party. Furthermore, for each dialog that is established, we add an edge between the two persons having this conversation if it does not yet exist. If the party were to go on infinitely [Ada82], the graph would be connected after a while, after a longer time, it will finally be a clique, i.e. a graph where every node is directly connected to each other node. The growth of the graph follows a stochastic process and the realization of a concrete configuration of the graph is the realization of a random graph.

Two formulations have been developed for the formation of such a graph. Bollobas [Bol01] introduced the concept of the $G_{n,p}$ random graph. Formally, the graph consists of n vertices. For each of the $\binom{n}{2}$ possible edges, imagine a coin being tossed that with probability p lands heads up and thus leads to the edge being added. With probability $1-p$ the edge is not added to the graph. Each graph thus created is one of $2^{\binom{n}{2}}$ possible realizations of the $G_{n,p}$ graph.

Asymptotically, this is equivalent to the $G_{n,N}$ formulation of a random graph where a set of N edges is chosen from all $\binom{n}{2}$ possible edges under the restriction that every possible graph with N edges and n nodes has the same probability of being created. Alternatively, a $G_{n,N}$ graph [ER57] can be created by iteratively adding edges from a uniform distribution over all possible edges not yet assigned until N edges are present. The equivalence of $G_{n,p}$ and $G_{n,N}$ for very large graphs follows when p and N are chosen such that $\binom{n}{2}p = N$ since the first is the expected number of edges in the $G_{n,p}$ graph and the latter is the fixed number of edges in the $G_{n,N}$ graph.

With a growing number of edges a phase change behavior with respect to different properties of the graph like connectedness is observable. A phase change is marked by the rather abrupt transition from one state to the other. For the analysis of this effect on random graphs, it is useful to introduce the graph family $G_{n,p(n)}$ [Spe01], where the probability of an edge being added is expressed as a function of the number of nodes. It can then be shown that for many properties of the graph, a threshold function $t(n)$ for $p(n)$ exists such that, in the asymptotic view, if $p(n) \ll t(n)$, the property almost surely does not apply to the graph, while for $p(n) \gg t(n)$, it almost surely holds. Tab. 2.1 lists some of these properties along with their respective threshold function.

Table 2.1: Threshold functions $t(n)$ for graph properties

Threshold	Property of the Graph
n^{-2}	The graph has edges
$n^{-\frac{3}{2}}$	The graph has nodes with a degree of at least two
$n^{-1-\frac{1}{k}}$	The graph contains trees with $k+1$ vertices
n^{-1}	The graph contains triangles and cycles
$\frac{\ln n}{n}$	The graph is connected
$n^{-\frac{2}{k-1}}$	The graph has complete subgraphs with k nodes ($k \geq 3$)
$n^{-\frac{1}{2}} \ln^{\frac{1}{2}} n$	Each pair of the graph's nodes has a common neighbor

Let us consider the property of being connected a bit further. If $p(n) = \frac{\ln n}{n}$, the expected number of edges is $\binom{n}{2}\frac{\ln n}{n} = \frac{(n-1)\ln n}{2} \approx \frac{1}{2}n \ln n$. According to Erdös and Renyi [ER57], the probability of a graph with $\frac{1}{2}n \log n + cn$ edges and n vertices being connected is

$$\lim_{n\to\infty} P(G_{n,[\frac{1}{2}n\log n+cn]} \text{is connected}) = e^{-e^{-2c}} \tag{2.9}$$

By varying c, we obtain the function plotted in Fig. 2.1. The phase change is

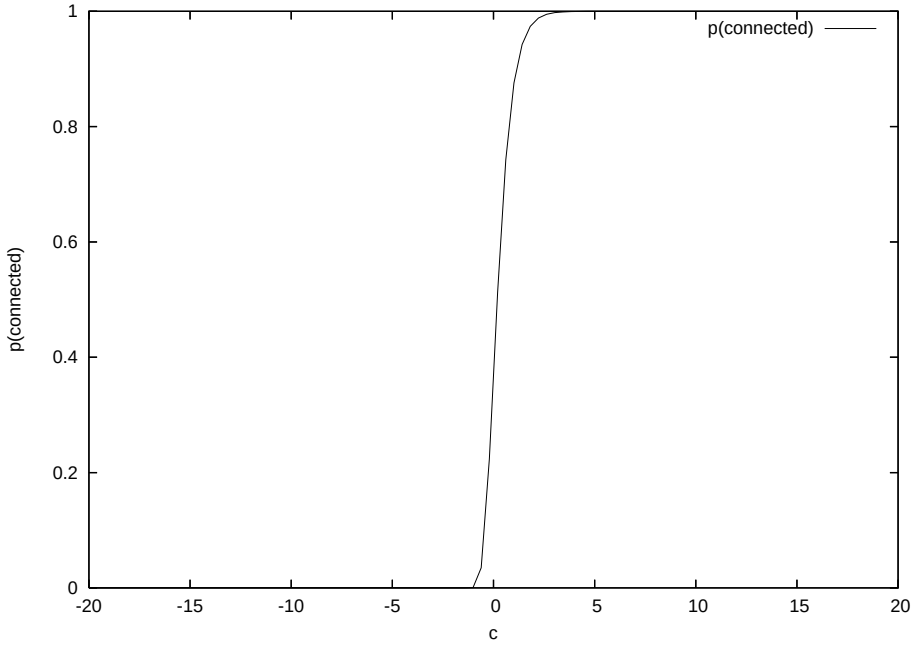


Figure 2.1: The probability of connectedness for $G_{n,[\frac{1}{2}n\log n+cn]}$ as a function of c [Fra03]

visible in the interval $[-1, 3]$, where the probability of the graph being connected goes up from practically zero to nearly one.

Random walks may also be considered as generators for such an incrementally growing graph, if for each edge in the similarity graph visited by the walk, an edge is added to the random graph between the appropriate place if it does not already exist. The difference, however, is the order in which the nodes and edges are visited: The order of the edges visited by the random walk is not as free as for the normal random graph since it depends on the underlying structure. Furthermore, using only one walk process, the graph always consists of exactly one component and many singletons. This can be overcome by using several walk processes, possibly with restrictions as introduced in chapter 3.

In that case, as stated in [Fra03], the most interesting moment for clustering purposes is when substructures in the graph start forming, but before the graph is connected. In this case, the substructures give information on the clusters present as will be detailed later when dealing with restricted random walks in chapter 3.

2.3 Cluster Algorithms

This section gives a short overview of the related literature on cluster algorithms in general and presents three algorithms, k-means, single pass, and hierarchical agglomerative clustering, in more detail since these will be used in the following section on dynamic clustering. The era of classification and clustering on computers began in 1957 with Sneath's seminal paper [Sne57] on the use of computers in taxonomy. It would be beyond the scope of this thesis to give an overview of the complete area of cluster algorithms. A more in-depth review can be found in [Fra03]. The book by Bock [Boc74] offers a very good introduction to the general ideas and principles of clustering and still is one of the standards in this field. Another good introduction is contained in the first chapters of the book by Kaufman and Rousseeuw [KR90], followed by the rather specific presentation of some cluster algorithms. Both Sokal [Sok77] and Duran and Odell [DO74] offer a broad review of cluster methods. For the practical application of clustering algorithms the paper by Dubes and Jain [DJ76] can be recommended. The works of Augustson and Minker [AM70], and Matula [Mat77] are focused especially on graph theoretic clustering methods. There have been attempts at defining general frameworks, for instance by Lance and Williams for hierarchical bottom-up cluster methods [LW67a, LW67b]. Jain et al. [JMF99] give a general overview over the subject of clustering, including such topics as fuzzy clustering [Bez81], clustering with artificial networks, and evolutionary clustering [GD03].

In general, the following requirements for a "good" clustering are considered in the literature [Fra03]:

- Stability: According to Godehardt and Jaworski [GJ02], the stability criterion comprises three aspects: First, the addition of new objects should not have a significant influence on the clustering, and the method must display insensitivity against reordering of the data as well as robustness against outliers. The first two aspects are especially important in the context of updateable cluster algorithms: The first is the discussion of plasticity versus stability of the clustering, i.e. the question of how strongly a clustering is adapted when new objects arrive. The second is even more critical for the quality of the clusters, as will be discussed in the following sections: Often, it is not possible for stream or incremental algorithms to order the data in a way that would create a good cluster quality. While not a specific problem of dynamic clustering, the question of how to treat outliers is also important since single outliers can change the cluster shape considerably when included in a cluster. It must thus be decided whether to discard outliers, keep them as singleton clusters or include them in the existing clusters.

- Comprehensibility: The resulting clustering must be comprehensible in the sense that the user must have the possibility to understand the steps that the algorithm has taken toward the solution. This is complemented by the next criterion,

- "Natural" clusters: Obviously, the clusters produced by an algorithm should be intuitive and compatible to the structure that is inherent in the data set. In an attempt to grasp the concept of natural clusters in terms of an objective quality measure, there has been a plethora of formal criteria for the quality of clusters, for instance by Halkidi et al. [HBV02b, HBV02a], Hagen and Kahng [HK92] or Pantel and Lin [PL02]. Halkidi et al. give a very good survey both of external criteria, when it is possible to compare the clustering with some a priori classification, and of relative measures, where two algorithms are compared. An example for the first type of measure is the Jaccard measure [Jac08] that computes the fraction of correctly classified object pairs over the number of pairs that are assigned to the same cluster either by the clustering, the original classification, or by both. A member of the relative class of criteria is the Dunn family of indices

$$D_{n_c} = \min_{i=1,\ldots,n_c} \left\{ \min_{j=i+1,\ldots,n_c} \left\{ \frac{d(c_i, c_j)}{\max_{k=1,\ldots,n_c} diam(c_k)} \right\} \right\} \tag{2.10}$$

  with n_c the number of clusters, $d(c_i, c_j) = \min_{x \in c_i, y \in c_j} d(x, y)$ the dissimilarity between the clusters c_i and c_j, and $diam(c_k) = \max_{x,y \in c_k} d(x, y)$. Hagen and Kahng have proposed the degree/separation measure that is described in section 2.5.4. The objective functions used in the algorithms by

Fisher [Fis58], Ward [War63], and others also represent an implicit quality criterion. Fisher, for instance uses the weighted quadratic distance or sum of squares from the cluster centroid

$$D = \sum_{i=1}^{K} w_i (a_i - \bar{a}_i)^2 \tag{2.11}$$

where K is the number of objects, w_i is the weight of object i, a_i is some numerical measure assigned to i – for instance the position in a metric space – and $\bar{a}_i$ is the arithmetic mean of the numerical measures of all objects assigned to the same cluster as i. It is then of course desirable to minimize this measure. However, in spite of all these attempts, it should be noted that the naturalness criterion is still hard to grasp formally.

- Efficiency: It is clear that even clusters of high quality are useless if their computation takes too much time, i.e. if the results are not available when they are needed. Consequently, the computational complexity of an algorithm should always be borne in mind, especially for large data sets. Many solutions that were proposed are NP-hard and as such hardly fit for real applications. But sometimes approximations can be found, for instance by a more efficient randomized algorithm as discussed in section 2.5.

- Shape independence: Optimally, a cluster algorithm should be able to identify clusters independently of their shape and relative size. However, as Jain et al. [JMF99] note, there is no universally applicable cluster algorithm. Rather, as mentioned in section 1.1, there exist algorithms that are specialized for clusters with certain shapes. For instance, k-means cluster algorithms are especially fit for the detection of spherical, compact clusters, whereas single linkage will reliably identify elongated clusters, but with a strong tendency towards bridging.

- Optionally, support for dynamic data sets as defined in the introduction: In addition to the above criteria, a dynamic algorithm should be able to cope efficiently with data sets of changing size, containing objects that may change their pairwise similarity over time. Especially, it is not deemed efficient to simply recompute the clustering using the whole data set. This requirement will be elaborated in section 2.4 for the evaluation of existing methods and will be the benchmark for the development of the algorithm discussed in this thesis.

Cluster algorithms can be classified according to various criteria that Good [Goo77] quite eloquently described in his “botryology of botryology”. The term

botryology – that never really caught on – is derived from the Greek word $\beta o \tau \rho \upsilon s$ that designates something that resembles a cluster of grapes. The more widely used criteria from his list, supplemented with those listed by Sneath and Sokal [SS73] are:

- disjunctive versus overlapping
- qualitative versus quantitative
- agglomerative versus divisive
- hierarchical versus nonhierarchical (partitional)
- local versus global criteria
- direct versus iterative solutions

In addition, the criteria

- deterministic versus stochastic (cf. section 2.5)
- hard versus fuzzy [Bez81]

should be mentioned.

The algorithm that is at the center of this thesis is either disjunctive or overlapping, depending on which cluster construction method is used, it is quantitative, hierarchical, uses local criteria, is direct, stochastic and hard. Due to the formulation of the process, it cannot be assigned the label agglomerative or divisive.

Before considering dynamic clustering in detail, let us review some static algorithms that will be referenced later in this thesis.

2.3.1 k-Means Clustering

k-means or k-medoids clustering is a simple iterative cluster algorithm developed by MacQueen [Mac67]. It finds clusters in a vector space for a predefined number k of cluster centers by iteratively adjusting the cluster centers and the association of the objects with these centers. Let $\mu_1, \ldots, \mu_k$ represent the cluster centers. In the case of k-means, they are computed as the average value of its cluster's members, for k-medoids, the μ_i are assigned the cluster member closest to this average. Usually, the algorithm is initialized with random vectors for the μ_i. In each iteration, first each of the objects is assigned to the cluster center it has the minimum distance to. Then the cluster centers are recalculated for the next iteration. The algorithm stops either after a predefined number of iterations or when the set of cluster centers does no longer change significantly.

An interesting preprocessing step for the k-means algorithm has been developed by Lin et al. [LVKG04]. It addresses two known problems for the algorithm: First, the running time of the k-means algorithm depends on the dimensionality of the input data, which constitutes a problem for time-series inputs. Second, the method being a hillclimbing algorithm, it may get stuck in local optima, depending on the initial distribution of the cluster centers.

Both issues can be alleviated by reducing the dimensionality of the input data set by using e.g. a Haar transformation; the authors claim that any multi-resolution decomposition such as discrete Fourier transformations is applicable, too. The idea behind a Haar transformation is to iteratively calculate the average of two adjacent data points at the next higher resolution; the Haar_0 coefficient is the global average. The transformation is lossless and thus completely reversible.

In order to avoid local minima, the first clustering is started on a low-resolution level. At this level, the probability of finding a local optimum is less pronounced than with high-resolution data. In further steps, the last solution obtained at a lower level is used as input for data of higher and higher resolution levels, using more and more Haar coefficients, until finally the original input data has been restored and is used for the clustering. The authors show that the running time in practice is lower than for the normal k-means algorithm and that the average quality is higher.

2.3.2 Single Pass Clustering

The term single pass clustering is applied for a class of clustering algorithms that cannot use random access on the data set but instead must compute cluster assignments as each object is introduced. One variant of this paradigm has been introduced by Deichsel [Dei78]. As each object is considered for clustering, it is either assigned to one of the existing clusters if its distance to one of the cluster members is below a predefined threshold, else it forms its own cluster.

The advantage of the single pass class of algorithms is their linear computational complexity. On the other hand, the quality of the resulting clusters is not optimal since every object can only be considered once, and before knowledge of the layout for the rest of the data set is present. Suboptimal decisions that were made at an early stage cannot be reverted. Furthermore, the results of such an algorithm depend on the order in which the objects to be clustered arrive. If, for example, the first objects are just outside each others' cluster radius, they are assigned to different clusters, even if later data points lying between these initial objects suggest that there should be one large cluster in the region instead of many small ones.

2.3.3 Hierarchical Agglomerative Clustering

The term Hierarchical Agglomerative Clustering (HAC) [War63] describes a general framework for obtaining a hierarchy of clusters as is often the requirement in information retrieval. The general idea is to start with a set of singleton clusters that form the leaves of the dendrogram. In each step of the algorithm, the two clusters that are closest to each other with respect to the metric used are merged, until there is only one cluster left: The root node of the dendrogram.

Usually, the metric is one of the following: Smallest distance between member objects of the clusters in question (single linkage), average distance (average linkage) or largest distance (complete linkage), diameter (largest intra-cluster distance of the merged cluster) or radius (the size of the sphere enclosing the cluster) of the clusters.

2.4 Dynamic Clustering

After the general overview of the last section, this section presents the approaches proposed for clustering dynamic data. The designation dynamic data can mean two things: First, it may be the size of the object set that increases or decreases over time as new members join or old members leave the set. Second, the members themselves may change their relation, i.e. distance or similarity, to other members, thus necessitating an adaptation of the cluster structure. A problem that occurs in this context is that of the stability/plasticity dilemma [DHS01]. On the one hand, the clustering should be able to integrate new objects as accurately as possible in the cluster structure: The clustering should be plastic. On the other hand, such a good integration may entail considerable changes in the cluster structure due to the insertion of single objects which means that the clusters are not stable. The conflict between plasticity and stability should be kept in mind when examining the algorithms reviewed here.

All algorithms compiled here can cope with at least one of the scenarios described above, where the insertion of nodes seems to be by far the most common problem – at least judging from the respective number of publications. Optimally, a cluster update algorithm should be able to cope with insertions, deletions, and changing similarities while maintaining the quality that would result from a renewed execution of the algorithm on the updated data set.

Another field of research where the problem of dynamic data sets is often encountered is the domain of data mining. Solutions were for instance proposed by Wang [Wan97] who used dynamic suffix trees to detect recurring patterns in sequential data (strings), given minimal values for support and confidence, or Parthasarathy et al. [PZOD99] who used the relations between subsequences and

sequences in data streams to quickly detect frequent sequences. These two algorithms will not be detailed further here since they rather belong to the domain of association mining than to clustering.

Section 2.4.1 contains an overview of algorithms that can handle growing and, in some cases, shrinking data sets. It starts with an overview of this algorithm class before presenting the different approaches published in this field. The section ends with an evaluation of the algorithms' capabilities in the context of dynamic data sets in the broader sense. The next section, 2.4.2, presents algorithms that are also capable of integrating changing similarities, for instance for the purpose of access pattern optimization in object-oriented data bases, followed by a summarizing evaluation. Finally, section 2.4.3 gives an overview about a domain of research that is currently quite active: mobile, wireless and ad-hoc networks. This class of algorithms naturally faces both dynamic object set sizes and moving objects; the methods are often used for finding optimal routes for the network. The important difference that discriminates algorithms for mobile scenarios from the ones described in section 2.4.2 is that they usually only have a binary similarity or distance measure (two nodes can directly reach each other or not). Furthermore, given that the execution environment for the algorithm is usually a mobile device with limited battery power, limited computational resources and limited communication bandwidth, the clustering algorithms have to make compromises between quality and resource usage, as will be discussed in the summary of section 2.4.3.

2.4.1 Integration of New Objects into an Existing Clustering

The typical application scenario in this problem class is the sequential appearance of objects for example in data streams. Clusters have to be constructed before the object set is complete. Application examples include environment detection for autonomous agents [MMR96] and especially stream clustering [Bar02].

In general, these algorithms are not capable of dealing with changing similarities or distances, but if the algorithm also supports the removal of objects, it is possible to integrate this scenario. For instance, if objects move with respect to each other, the moved objects can be first removed from the data set and reinserted at their new position. Whether this is efficient or produces clusters of the desired quality is a different question.

Cover-Coefficient-Based Incremental Clustering Methodology

One of the first publications in the area of dynamic clustering is the article of Can and Ozkarahan [CO87]. They stipulate the following requirements for dynamic clustering in an information retrieval environment that overlap with the general requirements for clusters given in section 2.3:

1. Stability: The appearance of new objects should not lead to substantial changes in the cluster structure. Equally, small errors in the determination of similarities e.g. from a textual description should not affect the clustering.

2. Order independence: The order in which the objects are presented to the algorithm should not have an influence on the cluster structure.

3. Well-definedness: The algorithm should either produce a single classification, i.e. clustering, or a small set of compatible classifications when clustering a given data set.

4. Uniformity of distribution of the documents in the clusters.

5. Efficiency: The addition – and possibly removal – of objects should be efficient and practical

6. Optimality for retrieval: The resulting clustering should allow an efficient and effective retrieval procedure.

The algorithm developed by Can et al. [CO87, CO89, CD90, Can93, CFSF95] was motivated by a typical information retrieval (IR) problem: Given m documents described by n terms, find groups of similar documents. The input data is given as a feature matrix $D_{m\times n}$ where the entry d_{ij} is either a binary variable that denotes whether document i is described by term j, or it contains the weight of term j in document i. In the following, binary entries will be assumed. The reciprocal of the sum of the i-th row of D is denoted by α_i, the reciprocal of the sum of its k-th column by β_k.

The cover coefficient matrix $C_{m\times m}$ describes for every document how well it is described by each other document. Its elements are calculated as

$$c_{ij} = \alpha_i \sum_{k=1}^{n} \beta_k d_{ik} d_{jk} \tag{2.12}$$

c_{ij} denotes how well i is covered by j. Due to the formulation of the c_{ij}, the row sums of C are equal to one. In addition to the coverage by other documents, the uniqueness or decoupling coefficient $\delta_i = c_{ii}$ represents the self-coverage of document i. The lower δ_i, the more i is coupled with other documents. The coupling coefficient of a document i is the row sum of the off-diagonal entries $\psi_i = 1 - \delta_i$. The coupling coefficient [sic] for the whole data base $\delta = \sum_{i=1}^{m} \frac{\delta_i}{m}$ is a good criterion for the appropriate number of clusters – note that despite the name, the coefficient has nothing to do with correlations or linear interactions.

The authors suggest to generate $n_c = \delta m$ clusters. For each document, the cluster seed power

$$p_i = \delta_i \psi_i \sum_{j=1}^{n} d_{ij} \tag{2.13}$$

determines its fitness to be a cluster center or seed. The n_c documents with the highest cluster seed power are selected as cluster seeds for the initial clustering. The other documents are then assigned to the cluster seed that covers them best according to c_{ij}.

For the cluster maintenance, the following algorithm is used. D_f is the set of documents that have to be (re)clustered:

1. Compute the p_i for all documents
2. Determine the cluster seeds of the new data set
3. If an old cluster seed is no longer a seed, add the documents from its cluster to D_f
4. If a non-seed becomes a seed, add all documents from the same cluster to D_f
5. Assign the members of D_f to the new cluster seeds as above

The overall complexity of this approach is given as $O(3t + m \log m) + O(n_{cm}(|D_f| - n_{cr})x_d)$ where t is the number of non-zero entries in the feature matrix D, n_{cm} is the number of cluster centers in the updated data set, n_{cr} the number of seeds in the set of documents to be (re)clustered and x_d is the average number of terms per document. The algorithm is designed for growing data sets, but could also be used when objects are removed. The quality of the clusters is maintained during the updates at the cost of a relatively high complexity.

Dynamic Clustering for Time Incremental Data

Another early work in the domain of incremental clustering is the article by Chaudhuri [Cha94] who systematically listed the possible actions upon arrival of new objects:

1. Absorption: A new object p is "sufficiently" close to an existing cluster in order to be considered as member. Distance in this case is defined in the sense of single linkage, i.e. as the minimum distance between p and one of the members of the respective cluster. Let C_i be the nearest cluster to p. Chaudhuri suggests that the distance of the object p to the closest cluster

member $q_0 \in C_i$ should approximately correspond to the average distance of q_0 to its m closest neighbors in C_i where m is a predefined constant.

2. Merging of clusters: If, during the steps described for case 1, two clusters C_i and C_j have reduced their distance, the two are merged. The distance between C_i and C_j is defined here as the minimum distance between one point $p \in C_i$ and another point $q \in C_j$:

$$d(C_i, C_j) = \min_{p \in C_i, q \in C_j} d(p, q) \tag{2.14}$$

   This case can only occur if at least one of the clusters has grown due to the absorption of one or several new objects and thus decreased its distance to some of the other clusters. The clusters are merged if their characteristics in the vicinity of p and q correspond. As a measure for the correspondence, the share of closest neighbors of p that come from C_j and vice versa is used. If it is close to $\frac{1}{2}$, Chaudhuri takes this as an indicator that the clusters' characteristics correspond.

3. New cluster formation: For the decision whether leftover objects from step 1 should form a new cluster or be considered as outliers, Chaudhuri proposes the use of a minimum spanning tree that is constructed over the set of objects not assigned to a cluster. For these objects, the average distance d_0 to their nearest neighbor in one of the existing clusters is computed, and all edges with an associated distance greater than αd_0, $\alpha > 1$ are discarded in order to obtain a threshold graph. Those subtrees with more than m_0 nodes form the new clusters, m_0 being a predefined threshold

4. Outliers: All objects not associated with a cluster in one of the preceding steps are considered as outliers

5. Split of clusters: At regular intervals, all clusters that have grown due to absorption or merging are considered for a split. First, the local densities are calculated for each object. If μ_l and μ_h are, respectively, the minimum and maximum local densities, all objects with a density less than $\mu_l + \beta(\mu_h - \mu_l)$ are discarded, where β is a given constant. Then, a minimum spanning tree is constructed. If the maximum distance associated with an edge is much higher than the average distance, clusters should be split at the edge with the highest weight, but only if its removal generates two subtrees with a "reasonable" number of nodes – unfortunately, the paper does not detail the question what a reasonable number of objects is for a cluster.

As can be seen, the actions here are triggered by criteria that are formulated in a vague way and must be concretized in order to be operationally usable.

Incremental Conceptual Clustering

Algorithms from the field of incremental conceptual clustering [Fis87, GLF89, Nev95] are among the earliest methods capable of integrating new objects. The methods are motivated by human learning processes vis-à-vis a set of yet unknown objects that are presented sequentially. In the general case, the knowledge of the structure inherent in the data set is represented by a decision tree where every node represents one or several criteria according to which the objects in its sub-hierarchies can be classified. While the static algorithms separate between a learning phase for the construction of the tree and the actual classification task, the incremental algorithms surveyed by Gennari et al. [GLF89] integrate both functions: Each object that is presented to the algorithm may influence the structure of the tree.

Gennari et al. [GLF89] propose to use an incremental hill climbing learning algorithm in order to cope with the dynamics of the object set and to reduce the memory requirements. In this case, the landscape the hill climber must cross is the space of all concept hierarchies, the altitude or quality of a single solution being determined according to the fit between the data presented so far and the hierarchy. Contrarily to static hill climbing methods, the hill climber in the context of incremental conceptual clustering is confronted with a changing "landscape" in every step. As a consequence, the algorithms may be order-dependent as the authors note in their survey.

Influence of Instance Ordering on the Clustering Quality

For most incremental algorithms, the order in which new objects arrive is crucial (cf. [GLF89]). Some of them deal with this problem by reorganizing the clusters with either local or global strategies. While local strategies are cheap to implement, it cannot be guaranteed that they correctly reflect global changes in the data. Global strategies, on the other hand, do not have this problem, but they are usually computationally much more complex than their local counterparts.

For algorithms without these reorganization facilities, it has been shown [FXZ92] that the best cluster results are achieved when the first objects presented to the algorithm are as dissimilar as possible, thus defining the regions in which objects can be found. In this case, the initial clusters can be expected to cover a large part of these regions. If, on the contrary, very similar objects are encountered at the beginning, the initial cluster centers are positioned close to each other. As a consequence, they must be repositioned when remote objects are presented to the algorithm. This increases the probability of the algorithm sticking to local optima.

Roure and Talavera [RT98] have therefore suggested the "not-yet" strategy for

various clustering algorithms, where each insertion is evaluated in terms of the algorithm's cluster quality criterion. If the expected utility, i.e. the confidence of adding the new object, is above a given threshold, it is inserted, else it is stored in a buffer to be inserted later on. The idea has a certain similarity to simulated annealing (cf. section 2.5.3) and has been shown by the authors to considerably reduce the influence of a bad instance ordering.

A similar principle has been proposed as the leader algorithm [Har75]: Given a vigilance parameter θ, each new object is either placed in a cluster if it lies within a radius of θ around its cluster center or it starts its own cluster if it is too remote from all existing clusters. Prehn and Sommer [PS06] calculate this vigilance parameter by iteratively clustering the object set with decreasing θ until the classification error falls below a predefined threshold. Clearly, the lower the vigilance parameter, the better the fit, but the lower the generalization capabilities of the clustering. The algorithm allows for ellipsoidal clusters, has a reduced sensitivity to instance ordering problems and runs in linear time of the number of clusters, but can only handle insertions.

Incremental Clustering and Dynamic Information Retrieval

In the context of information retrieval, Charikar et al. [CCFM97] have analyzed the extension of hierarchical agglomerative clustering to the dynamic case. Their goal is to maintain a predefined number of clusters with minimal diameter where the diameter of a cluster is defined as the maximum inter-object distance in the cluster. As a quality measure, the authors suggest the performance ratio that uses the optimal – i.e. static – clustering as a benchmark. The performance ratio is defined as the ratio of the maximum cluster diameter over all update sequences to the maximal cluster diameter of the optimal clustering.

Additionally, the dual problem of maintaining a fixed cluster diameter with a minimal number of clusters is discussed. The authors study three algorithms: a greedy algorithm with two alternative selection criteria, the doubling and the clique algorithm. The greedy algorithms work by merging either the clusters whose centers are closest (center-greedy) or those that minimize the diameter of the merged cluster (diameter-greedy). The two latter algorithms encompass several phases i that each consist of two stages: The first stage serves to shrink the initial number of $k+1$ clusters by merging, while new elements are added in the second stage. In the case of the doubling algorithm, the nearest clusters are merged in such a way that the radius, i.e. the maximum distance of each cluster member p from the cluster center c_j, $\max_{p \in C_j} d(c_j, p)$ does not exceed αd_i, where α is a predefined constant and d_i denotes a lower bound on the optimal cluster diameter (OPT) for the phase. Furthermore, the distance between two clusters must be $d(c_j, c_l) \geq d_i$ and $d_i \leq$ OPT. In the second phase, the new objects are inserted

into a cluster if it is possible without increasing the diameter of the cluster and given the respective restrictions for the cluster diameters. If this is not possible, a cluster is added that contains the new objects as only members. The second phase runs until the number of clusters exceeds k. The authors show that the performance ratio of the algorithm is 8. In other words, the cluster diameter for the most disadvantageous update sequence is smaller than eight times the diameter of the clusters in an optimal clustering of the same data set.

The clique algorithm works in a similar manner. In stage one, initially the following conditions hold: The radius of each cluster is at most $2d_i$, its diameter at most $3d_i$, $d_i \leq$ OPT. In the merging stage, a threshold graph G of the cluster centers is constructed with threshold $2d_i$. The cliques in G are then considered as new clusters. New objects are either added to existing clusters when the resulting cluster has a diameter of no more than $2d_i$, or they establish a new cluster. A new merging phase begins when the number of clusters exceeds $k + 1$. Although the performance ratio of the clique algorithm is 6 and thus better than that of the doubling algorithm, the problem is that computing cliques is NP-hard and therefore not really practicable for dynamic clustering.

Both variants can only handle insertions. In addition, the results of the update procedure are not equivalent to those of the original algorithm.

The greedy and the doubling algorithm have for instance been used in the clustering of user profiles for web agents by Somlo and Howe [SH01]. They showed that for this application, the doubling algorithm was superior to the greedy algorithm.

BIRCH

The algorithm BIRCH [ZRL96] has been primarily developed by Zhang et al. as a memory-saving algorithm with several passes, but can also be used as single pass variant; in that case, the quality obtained with several passes is of course not attainable. The goal of the algorithm is to reduce large data sets by representing densely populated (“crowded”) areas of a metric space as clusters that should fit into a single memory page and by discarding outliers.

The cluster concept relies on six concepts: centroid, radius, and diameter of a cluster, the clustering feature (CF), a distance measure, and the CF tree. For a cluster consisting of N elements $X_1, \ldots, X_N$, the centroid is given by

$$X_0 = \frac{\sum_{i=1}^{N} X_i}{N} \tag{2.15}$$

The radius is defined as

$$R = \left(\frac{\sum_{i=1}^{N} (X_i - X_0)^2}{N} \right)^{\frac{1}{2}} \tag{2.16}$$

and the diameter as

$$D = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (X_i - X_j)^2}{N(N-1)} \right)^{\frac{1}{2}} \tag{2.17}$$

In order to compute these characteristics of a cluster, the clustering feature CF stores for every cluster the tuple $(N, \sum_{i=1}^{N} X_i, \sum_{i=1}^{N} X_i^2)$ that is sufficient to calculate centroid, radius and diameter. Different measures are applicable for the inter-cluster distance, for instance the Euclidean or Manhattan/city-block distance (definition 1.1.8). Finally, a CF tree is a height-balanced tree whose dimensions are guided by the following parameters: The branching factor B determines the maximum number of child entries in a non-leaf node, the parameter L the maximum number of entries in a leaf node and the threshold factor T is the upper limit for either the diameter or radius of the cluster composed of the elements in one entry of a leaf node. If $X = \{X_1, \ldots, X_N\}$ is the set of all objects to be clustered, an entry $X_c \subseteq X$ of a leaf node thus has the form $X_c = \{X_{c_1}, \ldots, X_{c_k} | R(X_c) \leq T\}$ or $X_c = \{X_{c_1}, \ldots, X_{c_k} | D(X_c) \leq T\}$, respectively. A non-leaf node consists of the clustering features of its child nodes or subclusters and pointers to the children. Each entry of a leaf node contains several objects that form a subcluster with maximum diameter or radius T; this is called the threshold condition. Zhang et al. do not give further information on the height balance of the tree.

A new element is inserted into a CF tree by descending the tree, always choosing the entry that is closest to the new object until a leaf node is reached. The new element is inserted into the leaf's closest entry, if the growing cluster still satisfies the threshold condition. Otherwise, a new leaf entry is created for the new element. If this is not possible because the leaf already has L entries, the leaf node is split. Finally, the CF information on the path from the modified leaf or leaves to the tree root is updated.

The BIRCH algorithm consists of four phases, of which the first is most interesting here, because it generates a clustering while scanning the data only once (first pass). Phase two optionally reduces the size of the tree, phases three and four serve to improve an existing clustering, but need a further scan of the data set, which is why they cannot be used sensibly on stream or dynamic data.

Phase one builds the CF tree while scanning the data set. Zhang et al. claim that at any point in time during the CF tree construction, the tree represents a good clustering of the data processed so far. As a consequence, the algorithm is able, using only the first phase, to work on dynamically growing data sets.

The static algorithm was evaluated against the CLARANS cluster algorithm [KR90] which it outperformed both with respect to the cluster quality and the run time. The dynamic version copes only with new objects and has an inferior quality compared to the static one which would necessitate a reclustering.

Star Clusters

In contrast to algorithms that demand the number of clusters to be fixed a priori, the method developed by Aslam et al. [APR97, APR98, APR99] is claimed to find a "natural" number of clusters. As a result, the authors state that this procedure guarantees – contrarily to algorithms with a fixed number of clusters – a lower bound on the similarities inside a cluster. Similarity in this case is defined as the cosine measure (cf. definition 1.1.11). The algorithm is inspired by clique clusters, but since their computation is NP-hard, the authors instead employ dense areas of the graph with a star-like structure as clusters.

A high intra-cluster similarity is achieved by thresholding the similarity graph $G = (V, E, w)$, i.e. by removing all edges with a weight lower than a predefined threshold σ. While the thresholding is sufficient to guarantee minimum similarity between cluster members for clique clusters, this is not automatically the case for star-shaped clusters. For two cluster satellites – i.e. non-cluster centers – the expected similarity is $\cos\alpha_1 + \cos\alpha_2 + \cos\theta \sin\alpha_1 \sin\alpha_2$ where α_1 and α_2 are the respective angles between the satellites and the center and θ is the dihedral angle between the planes respectively formed by the center and one of the satellites.

The algorithm in its static version is a greedy cover algorithm for finding star graphs as follows:

1. Set $G_\sigma = (V, E_\sigma)$, $E_\sigma = \{e \in E | w(e) \geq \sigma\}$
2. Set the status of all vertices in G_σ to unmarked
3. Calculate the degree of all vertices
4. While unmarked nodes exist:
   (a) Let the unmarked node with the highest degree be a new star cluster center.
   (b) Add all adjacent nodes as satellites and mark the satellites and the cluster center
5. Represent each cluster by the object that is designated as its center

The resulting clustering is not unique since it depends on the order in which the objects arrive in the sorted list used for the loop. Per construction, it fulfills the correctness criterion stipulated by the authors: No two cluster centers are adjacent and each object has at least one adjacent cluster center.

The dynamic, online version of the algorithm is one of the few that explicitly treat the removal of objects. When inserting an object, the following cases may occur:

- If no star centers are adjacent to the new node, it becomes a star center itself.
- If at least one of the adjacent star centers has a higher degree than the new node, it is assigned to the respective cluster or clusters.
- If all adjacent star centers have a lower degree than the new node, their clusters are disbanded and all nodes not belonging to a cluster are reclustered.
- If the insertion causes a satellite to obtain a higher degree than its star center, the respective cluster is disbanded and all nodes not belonging to a cluster are reclustered.

Equally, the removal of a node may cause the following situations:

- If the node is a satellite, it is removed.
- If the node is a cluster center, the cluster is removed and the nodes not assigned to a cluster are reclustered.
- If the removal causes a star center to obtain a lower degree than one of its satellites, the members of this star must be reclustered.

The authors give a complexity of $O(n^2 \log^2 n)$ for the insertion of n nodes into an (initially empty) graph and for the removal of n nodes from an (n-vertex) graph.

Incremental Gravitational Clustering

The general idea of clustering using analogies to gravity has been proposed by Wright [Wri77], Kundu [Kun99], and Gomez et al. [GDN03]. Chen et al. [CHO02, CHO05] propose the GRACE algorithm for clustering that is based on a model for the movement of water drops in a space craft. The two influencing factors for a water drop in this environment are gravitational forces and friction generated by contact with the air molecules. The velocity of a particle j is given as

$$v_j = \sqrt{\frac{||\sum_{node\, n_i} F_{g_i}||}{C_r}} \tag{2.18}$$

where C_r is the air resistance and F_{g_i} is the force i experiences from all other particles. In the case of two particles i and j its amount is given as

$$||F_{g_i}|| = C_g \frac{(\text{mass of } n_i)(\text{mass of } n_j)}{\text{distance}^k(n_i, n_j)} \tag{2.19}$$

with k=2 "in the physical world" and C_g a "coefficient" [sic]. A further interpretation of the two terms is not given. The mass of a cluster is defined as the number of objects it contains.

Using these forces, the positions of all particles are iteratively updated. When two drops meet, i.e. when their centers' distance is smaller than the sum of their radii, they merge and continue together. The result is a cluster hierarchy formed over the course of the algorithm's execution. A cluster at a given level is characterized by a sphere around its centroid; the radius of the sphere is the maximum distance between the centroid and the cluster members. Obviously the spheres may overlap, even if the resulting clusters are disjunctive in terms of the objects they contain.

Based on the GRACE algorithm, the authors have developed the GRIN algorithm. The idea is to use the GRACE algorithm to obtain an initial clustering for a randomly selected subset of constant size and to assign the remaining original objects as well as new objects to these clusters. If the resulting clusters, once again described by their centroid and radius, have a sufficient number of members, they are tested for spherical shape: The distribution of their members is χ^2-tested for uniformity. Clusters too small to be sensibly tested are assumed to be spherical. These clusters together with the clusters that pass the test for their shape are considered for the next stage. All clusters that

1. are of spherical shape,
2. have – if any – only spherical descendants, and
3. have a parent that does not fulfill the two above conditions

are inserted as so-called leaf clusters into a tentative dendrogram. Their subtrees are flattened, i.e. all elements that belong to the subtree are attached as direct children of the leaf cluster. Optionally, outliers can be removed and the dendrogram can be rebuilt in their absence. This step is especially recommended if the outliers have a strong influence on the clustering.

With the dendrogram completed, the remaining data points as well as the ones arriving at a later point of time can be integrated. If a new datum falls into the sphere of a cluster, it is integrated. If it falls into the spheres of several clusters, the cluster exerting the highest gravitational force, determined from the cluster's weight and distance, is selected to receive the object. If the new element lies outside every cluster, the algorithm attempts to enlarge the cluster exercising the highest gravitational force on the new element such that it contains the new element and still passes the test for its spherical shape. In later versions of the algorithm, a step has been added that checks each time a new element is added to a cluster whether the cluster in question should be split in light of the latest

addition. Finally, if none of the above steps succeeds, the new object is put into the tentative outlier buffer.

When a threshold on the number of objects in the tentative outlier buffer is reached, the object set has to be reclustered using GRACE as described above using the old leaf clusters and the contents of the tentative outlier buffer as input objects.

The time complexity for the first, static phase is $O(n^2)$ for n objects and constant if the dendrogram is constructed using a fixed-size sample. The update phase has a complexity of $O(n)$ if the dendrogram cannot grow infinitely and of $O(n^2)$ if it does. The authors show that the results are superior to the BIRCH [ZRL96] algorithm for the data sets chosen in the paper. The algorithm only handles insertions, and the requirement of regular reclusterings is part of its design.

Incremental DBSCAN

Ester et al. [EKS+98] have proposed an incremental version of the DBSCAN (Density-Based Spacial Clustering of Applications with Noise [SEKX98]) algorithm in a data warehouse context, supporting both insertions and deletions. The general idea is to define clusters as regions with high local density. An object p is said to be directly density-reachable by another object q with respect to a given parameter Eps if p lies within a radius of Eps of q and the circle $N_{Eps}(q)$ with this radius Eps contains at least $Minpts$ other objects. Density-reachability is given if there is a chain of directly density-reachable objects from q to p. Finally, p and q are density-connected if there exists an intermediary object o such that p and q are both density-reachable from o. For a cluster, the following two conditions hold for clusters with at least $Minpts$ members:

- The clusters are maximal, i.e. with p, all objects density-reachable from p are also in a cluster, and
- The cluster is connected, i.e. all objects inside a cluster are density-connected.

Clusters are initially constructed by iterating over all nodes, recursively retrieving the Eps-neighborhoods and thus identifying density-connected regions as clusters. Cluster members are either core objects if their Eps-neighborhood contains at least $Minpts$ other objects, or border objects otherwise. Non-cluster members, i.e. objects not density-reachable from a core object and having a sparse Eps-neighborhood are classified as outliers or noise. The algorithm works iteratively by considering yet unclassified objects and either constructing a cluster around them if they are core objects or designating them as noise if their density is not

sufficient. Note that objects categorized as noise may later become border objects if they are density-reachable from a core object.

Due to the fact that cluster construction is density-based, the consequences of an update are locally contained inside a circle with radius $2Eps$ around the inserted or deleted object p. Therefore the set of affected objects is given as the objects in $N_{Eps}(p)$ together with any object q in the $2Eps$-neighborhood that is density-reachable from an object in $N_{Eps}(p)$, since the insertion or deletion may have established or destroyed a chain of density-reachable objects, thus affecting the density-connectedness of q. For these, the clustering must be updated accordingly.

In the case of an insertion, new density-connections may be established, but none will be lost. The set of concerned objects in this case contains all objects that have the core property after the insertion and are in the Eps-neighborhood of those objects that have acquired the core property due to the update. The cases distinguishable are thus

- Noise: The set of concerned nodes is empty, p is an outlier.
- Creation of a new cluster: Through the insertion of p some previous noise objects have become core objects that solicit the creation of a new cluster also including p.
- Absorption: All concerned objects are core objects contained in one preexisting cluster; p is added to that cluster.
- Merge: The concerned objects are core objects from at least two different clusters, in this case, the clusters are merged, and p is included in the united cluster.

For the deletion, those objects have to be taken into account that are still core objects after the deletion, but have in their Eps-neighborhood at least one object that has lost the core property due to the update. Here the possible actions include:

- Removal: If no further objects are affected by the deletion, it is sufficient to delete p from the object set.
- Reduction: If all concerned objects are directly density-reachable from each other, the cluster is preserved.
- Split: If the objects are not directly density-reachable and they are not density-connected by other objects from the same cluster, the cluster must be split in such a way that the cluster conditions given above are met again.

The algorithm can cope with insertions and deletions. Furthermore, as the authors have shown, the advantage of this incremental formulation of DBSCAN is that it produces the same results as the static version executed on the changed object set. Thus, there is no degradation over time that would make an eventual complete reclustering necessary.

An extension of (incremental) DBSCAN, (inc)OPTICS has been presented by Kriegel et al. [KKG03]. The algorithm uses the density-based principles of DBSCAN, but produces a hierarchical clustering.

Document Cluster Trees

In the context of web page classification Wong and Fu [WF00] have established the document cluster (DC) tree as a representation for web page clusters that can easily be updated. Borrowing concepts from the B^+ algorithm [YM98], it also supports the deletion of nodes. Each node in the tree represents either a document, a set of documents or a cluster. In the latter case, the entry is composed of at most B children containing the subclusters where B is a predefined value, the so-called branching factor.

When inserting a new web document, it is passed down the tree starting from the root node. In each step, the child it has the highest similarity to is selected as long as the similarity is above a given threshold. When no such child exists at a given step, the element is inserted as a new child in the current node. If in one of these cases the node created by this procedure has more than B children, it is split by choosing the pair of child nodes with the lowest pairwise similarity and using them as seeds for the newly created clusters.

Clustering with Cluster Similarity Histograms

Hammouda and Kamel [HK03] have proposed the similarity histogram-based overlapping clustering (SHC) algorithm that is claimed to guarantee a high degree of coherency for each cluster at all times. A cluster similarity histogram for a cluster is defined as having a number B of bins h_i that correspond to intervals $[s_{li}, s_{ui})$ with s_{li} and s_{ui} the lower and upper limit of bin h_i. Each bin is assigned the number of pairwise similarities inside the cluster that fall inside the respective interval. The idea in incremental cluster maintenance is to keep the distribution represented by the histogram as skewed to the right as possible when adding new objects. Given a similarity threshold S_T, the skewness is measured using the histogram ratio

$$HR_C = \frac{\sum_{i=T}^{B} h_i}{\sum_{j=1}^{B} h_j}$$

where T is the number of the bin that is associated with the similarity threshold S_T. A new object is either accepted when its addition to a cluster increases the histogram ratio or at least does not decrease it by more than a given $\epsilon > 0$ that is fixed in advance. Also, the histogram ratio is bounded by a lower threshold such that successive degradation by new objects cannot result in too low a ratio.

For the insertion of a new object, the SHC and ratio for all clusters are computed before and after an imaginary insertion. If the updated ratio for a cluster is better or if it is not worse by ϵ and above the threshold, the object is inserted. If an object cannot be assigned to one of the existing clusters, it forms its own cluster. The complexity for the insertion is $O(n^2)$. A disadvantage of this algorithm is that the clusters depend on the order of the newly arriving objects. The authors try to mitigate this effect by introducing a reassignment strategy: For each document, the improvement in the HR is saved that would result from its removal from the cluster. When a new object arrives and is inserted in a cluster, those nodes are candidates for removal whose deletion from the cluster would result in a higher HR than the current one. If one of them can be inserted into another cluster in such a way that the HR of that cluster is also improved, the respective node is reassigned.

Unfortunately, the authors do not detail the question of how to obtain an initial clustering. It is thus not clear whether the incremental algorithm is used starting from the first object or whether another method is employed to initialize the clusters.

The authors have tested the method on two small data sets containing web documents and have found the performance of the algorithm superior to HAC, single-pass and k-nearest neighbors.

Incremental Hierarchical Clustering

Ribert et al. [REL99], motivated by the high memory requirements for the clustering of large data sets with more than 10,000 objects, propose an incremental hierarchical clustering method both for dynamic data bases and for handling large data sets. The memory requirements are effectively reduced by starting with a small subset and iteratively adding the remaining objects to the clustering. They build upon the algorithmic framework for the Lance-Williams family of hierarchical cluster algorithms [LW67a] that encompasses single linkage, complete linkage and average linkage clustering. In their paper, the authors consider average linkage as underlying algorithm.

For the insertion, first the place in the dendrogram has to be found where the new element should be introduced. This is achieved via computing the regions of influence (ROI) for the clusters, i.e. the sphere around the cluster center that has a radius inferior to the distance to the closest cluster. Thus, if the new element

is in the ROI for a cluster, a new node is inserted above the cluster in question that merges the new element with the cluster. Afterwards, subtrees may have to be moved inside the dendrogram due to the fact that their distances may have changed. The search for these changes is propagated from the new position of the element upward to the root cluster. Two scenarios are possible: Two subtrees are closer than they were before the update – this is not possible for complete linkage – or they are further removed – this is not possible when using single linkage. The algorithm has only been evaluated in terms of its purported goal, the reduction of the memory requirement of the algorithm. The resulting reduction is indeed visible, the authors claim that the incremental version can handle seven times more elements for a given memory size than the conventional algorithm. The algorithm does not support deletions.

Another idea for incremental hierarchical clustering has been developed by Widyantoro et al. [WIY02]. It is based on cluster densities that are defined as the average distance of the cluster members to their respective nearest neighbors in the same cluster. The requirements for a cluster hierarchy are homogeneity and monotonicity. The first means that intra-cluster densities should be homogeneous. The second implies that the density of a child cluster is always higher than that of its parents. In this way a new element is inserted into the cluster tree where it least disrupts the two criteria, either by appending the new object to the child list of a node already present in the tree if the density lies in a predefined interval or by opening an intermediate cluster if the density lies between the intervals of parent and child cluster.

Only insertions are possible into the cluster tree. For the algorithm, the input ordering is not important. The authors have compared their algorithm with Fisher's COBWEB [Fis87] algorithm. Both algorithms show comparable performance if the ordering of the elements is random. If, contrarily, the ordering is bad, accuracy remains constant for the algorithm discussed, while it drops considerably for COBWEB.

Clustering Data Streams

Due to the large growth of storage capacities over the last decade, more and more data are available as data streams, i.e. as ever-growing data sets. Given the fact that in the extremal case, no data is ever discarded, it is quite a challenge to efficiently keep clusters derived from these data sets up to date. A general list of requirements for algorithms that cluster data arriving as streams has been given by Barbará [Bar02]. In general, when dealing with streams, it is assumed that the data that has arrived up to the current point in time is already included in the clustering. New data must be integrated with as little cost as possible. Therefore, the following criteria should be met by the algorithm:

1. Compactness of representation: Since the computation of the new clusters has to be efficient, the algorithm must offer a compact representation of each cluster. This is due to restrictions naturally imposed by the memory size of the computer. Furthermore, the list of cluster representations should optimally have a constant size; even a linear growth of the clusters' representation list is considered intolerable.

2. Fast incremental processing of new data points: In most cases, comparing a new point to all points in each cluster is not feasible. Thus the function should use the compact representation of the clusters. Furthermore, it should display a "good performance" in deciding about the membership of the new objects in the respective clusters.

3. Clear and fast identification of outliers: When trends in the data stream change, one of the signs is a higher number of outliers that cannot be fitted into the existing cluster model. In that case, the algorithm should be able to mark those outliers and to decide when to start a reclustering of the data set as stated by Barbará and Chen [BC01]. Depending on the application at hand, this can mean that either the whole data set is reclustered, or that a clustering containing only the newly arrived data is computed.

In general, the data cannot be accessed using random access – either because the data set is too large to fit into the main memory or because the original data stream is never stored on a disk, but is used to compute the clusters and discarded afterwards. Thus, in addition to the usual criteria for cluster algorithms like computational complexity and quality of the results, it is important to know how many linear passes a stream cluster algorithm needs to compute the clusters. Of course, if the data is never stored on a hard disk, only one pass is possible.

O'Callaghan et al. [OMM$^+$02, GMM$^+$03] have developed a k-median based stream clustering algorithm. It produces at most k clusters and relies on two criteria: The first is to minimize the sum of squared distances inside the cluster. Since, in terms of this criterion, singleton clusters are optimal if k is not fixed in advance, it is linearly combined with a second criterion, a cost function for the number of cluster centers to avoid the formation of too many clusters. The data is assumed to arrive in chunks that each can be clustered in main memory using a k-median variant. The cluster centers found for each chunk are added to the set of all cluster centers found in any prior iteration and are weighted with the size of their associated cluster in order to be clustered themselves. The algorithm cannot handle deletions and is dependent on the order in which the objects are presented.

The approach is enhanced in [COP03] where not only the current data chunk is used for k-median clustering, but also the result from previous iterations of the algorithm.

Gupta and Grossman [GG04] present GenIc, another single-pass algorithm that is inspired by the principles of evolutionary algorithms (cf. section 2.5.2) and only supports insertions. The population consists of the cluster centers c_i. As each data chunk arrives, the fitness of the cluster centers included in the current generation is measured as their ability to attract a new object p in this chunk. The successful cluster centers in this process are moved according to

$$c_i = \frac{(w_i c_i + p)}{w_i + 1}$$

where w_i is the number of objects currently attached to c_i. The term c_i appears in both sides of the equation in the original paper, although, on the right hand side it is assumed to represent the cluster center before the update, and after the update on the left hand side. At the end of each data chunk, the survival of each center is decided at random in accordance to its fitness. Centers that do not survive the selection are replaced by random points. In an evaluation conducted by the authors, the GenIc algorithm performance surpassed that of windowed k-means and was at least equivalent in terms of quality to the standard k-means algorithm on a synthetic data set.

A similar idea from the area of conceptual clustering was developed earlier by Rowland and Vesonder [RV87]. If a cluster center and a new object are very similar, the cluster center is replaced by a generalization of itself such that it is also able to represent the new object.

Aggarwal et al. [AHWY03] criticize that often single-pass algorithms are used for clustering data streams that ignore temporal trends. Instead, analyzing a long time span, historical data may prevail over current trends, which means that the most recent evolution in the stream is not registered by the user. As a remedy, the authors propose a two-phase approach incorporated in the CluStream algorithm that consists of an online and an offline component. The online component summarizes the cluster structure in so-called micro clusters that contain a condensed representation of the clusters at given times; the granularity of these clusters decreases with the age of the clusters. The offline component builds upon these micro-clusters to offer different time frames for the analysis of the data stream by combining clusters inside the time frame requested by the user. Since it can operate on the summaries provided by the online component, the authors claim that it can execute its computations efficiently. An evaluation against the BIRCH [ZRL96] and the stream algorithm from [OMM+02] showed both faster execution times and higher cluster accuracy for the CluStream algorithm.

Zhong [Zho05] uses the ideas behind Aggarwal's framework in order to cluster text streams with a stream variant of the spherical k-means algorithm and combines it with principles from machine learning. The algorithm does not only assign new documents to clusters using their cosine similarity (Eq. (1.12)) to the center

of the respective clusters. Following the winner-takes all principle [AKCM90] the center of the cluster in question is updated using a decreasing learning rate. This approach has some similarities to the online clustering presented by Duda et al. [DHS01] or simulated annealing (cf. section 2.5.3). In order to improve the running time of the algorithm, Zhong proposes to only sample a part of the objects for the learning of the cluster centers, especially at the beginning. The data is, as is the case with many previously shown algorithms, read in chunks, allowing a better adaption of the clusters by repeatedly visiting the newly arrived objects. The results on the data sets employed are superior to both the spherical k-means and the CLUTO algorithm [Kar, Kar03] that the author considers state of the art.

Other approaches also deal with noisy data streams, either by using an artificial immune system [NUCG03] or by density-based clustering [NUCG03, CEQZ06], but will not be detailed here.

All stream algorithms presented here only handle insertions and are dependent on the order of input.

Summary

There is quite a lot of literature on the subject of growing data sets. Of the 45 documents presented in this section, two are from the seventies, five from the eighties, and 17 from the nineties, where the subject of dynamic clustering really started to develop. The dynamics of this domain are also reflected in the fact that nearly half of the contributions in this section have been written after 2000. The methods reviewed here were motivated by a wide range of applications, from machine learning over information retrieval to the handling of data that arrives as stream instead of the usual static data sets. A problem that is frequently encountered is that of the sequence of the objects. Quite a few algorithms produce different results when the objects are presented in a different order. Because this contradicts the stability requirement for clusters, strategies for reordering data have been proposed in order to mitigate these effects, or criteria have been developed to determine the point at which a reclustering is necessary.

The removal of objects is explicitly supported only by three algorithms, star clusters [APR97, APR98, APR99], document trees [WF00], and incremental DBSCAN [EKS+98].

For the scenario that was outlined in the motivation, these algorithms are not satisfying since

1. changing similarities are not directly supported – although it is possible to handle these by removing and subsequently adding the objects in question,

2. the results of most update algorithms differ from those given by the original algorithm, and a reclustering may be necessary after some time,

3. the input order of the objects should not play a role for the results produced by the algorithm.

2.4.2 Handling Changing Similarities

There is actually quite little literature on the subject of objects that can change their pairwise similarities. There are some approaches based on incremental k-means or k-medoids clustering (e.g. [CKT06]), and the data base literature provides some further examples for this kind of dynamic clustering [MK94, BS96, DG96, DFR+01]. Finally, the mobile/wireless/ad hoc scenarios described in the next section, 2.4.3, offer some distributed dynamic cluster algorithms, but their quality is in most cases limited in favor of an easy computation and low communication overhead.

Incremental k-Means

An approach labeled "Evolutionary Clustering" has been put forward by Chakrabarti et al. [CKT06] – this is not to be confused with evolutionary or genetic algorithms as discussed in section 2.5.2. Rather, evolutionary in this context means that the clusters evolve over time as new data arrives. The objective is to find a balance between a reliable stability of the results over time and a truthful representation of the data at the current time.

Consequently, the quality of a clustering C_t at time t with respect to a matrix M_t containing similarities or distances depends on the snapshot quality $\text{sq}(C_t, M_t)$ and the history cost $\text{hc}(C_{t-1}, C_t)$, i.e. the distance to the clustering at $t-1$ as a weighted function, leading to the total quality of a clustering sequence of

$$\sum_{t=1}^{T} \text{sq}(C_t, M_t) - \text{cp} \sum_{t=1}^{T} \text{hc}(C_{t-1}, C_t) \tag{2.20}$$

where cp is the user-supplied weight for the history costs.

The method can be applied to different cluster algorithms, for instance k-means: At each step, the clustering is initialized using the clusters from the last time step – this is the basic incremental k-means clustering – and the usual iterations are executed. The snapshot quality is then given by

$$\text{sq}(C, M) = \sum_{x \in U} (1 - \min_{c \in C} ||c - x||) \tag{2.21}$$

where C is the set of cluster centroids and U is the set of objects to be clustered; both objects and centroids are represented as vectors. The same norm $||\cdot||$ must

be used for both the k-means and the update part of the algorithm. Equally, the history costs between two clusterings C and C' are defined by

$$\mathrm{hc}(C, C') = \min_{f:[k]\to[k]} ||c_i - c'_{f(i)}|| \tag{2.22}$$

with $f : [k] \to [k]$ being a function that maps the cluster centroids of C to those of C'. The authors stress that the determination of cp is important as it has a strong influence on the trade-off between stability in form of small history costs and plasticity in form of the snapshot quality. Due to the complete reclustering at each step, all kinds of changes are supported, but at high computational costs. The quality of the clusters is mainly determined by the underlying algorithm.

Object Data Bases and Clustering

With the advent of object-oriented data bases (OODB) came the demand for online cluster techniques that support the specifics of this type of data base. In the context of OODBs, clusters are sought that contain data often accessed together and that fit into a given amount of main or secondary memory in order to optimize access times. In an object-oriented data base, the data structure constituting an object may be stored in different locations due to the fact that these parts of the object are inherited or belong to another object referenced by the one currently under consideration. This leads to the requirement to support not only set-oriented operations, but also materialization and navigational access based ones: The materialization access takes place when an object is recalled and its parts must be localized and read from secondary storage. The navigational access is caused by the recursive retrieval of the object's complex components. A general review of dynamic clustering algorithms for object-oriented data bases can be found in the contributions by Darmont et al. [DFR$^+$01, DG96]. Although simple greedy algorithms like for instance CACTIS [HK89] do not perform well, the results discussed in [DFR$^+$01] show that it is not always the most complex algorithm that gives the best results in dynamic scenarios either. In this case, it is the DRO algorithm (Detection & Reclustering of Objects) that is claimed both to be relatively simple to implement and to return clusters of high quality.

The DRO algorithm uses statistics on objects as well as on pages. A page in this context is the *unit of transfer between disk and memory*. For each object j, its access frequency AF_j, i.e. the number of times it has been accessed, is stored as well as a binary attribute indicating whether it has been accessed at all. For a page, the number of times the page has been loaded – the load number – is recorded as well as the usage rate, i.e. the ratio of active data stored on the page to its total capacity. When an object is accessed, its access frequency is increased, the usage flag is set to true and the usage rate of the corresponding page or pages

is recomputed. When a page is loaded from disc, its load number is increased. In the case of object removal, the respective statistics are deleted. If an object is moved to another page during the clustering, its statistics are reset. The actual clustering is divided in four steps:

1. Determination of objects to be reclustered: If there exists more than one page that has been loaded more often than a given threshold $MinLT$, but has a usage rate lower than the threshold $MinUR$, the objects in these pages are scheduled for reclustering. This criterion aims at removing inactive objects from frequently accessed pages. If the relation between pages to be clustered and the overall number of pages used is sufficiently high, the next step is executed.

2. Clustering Setup: The input for this stage is the list of objects to be clustered, sorted in descending order of access frequency, as found by the previous stage. In the first phase of this stage, the references of each object i are evaluated up to a predefined depth $MaxD$ and the corresponding linked objects j are inserted into a list of references sorted in descending order according to their access frequency, but only if their dissimilarity rate $\frac{|AF_i - AF_j|}{\max(AF_i, AF_j)}$ to the starting object is below the threshold $MaxDR$. The list determines the order in which the objects are arranged on disk, in other words, the cluster for the object i. The lists are concatenated into a single list in the second phase, and in the third phase, the resemblance rate is computed as the fraction of objects in the proposed cluster that are not moved with respect to the current situation to the total number of objects in the respective cluster.

3. Physical Object Clustering: If the resemblance rate is below the threshold $MaxRR$, the objects are moved on disk according to the assembled list from stage 2. Furthermore, space no longer used by moved or deleted objects is released.

4. Statistics Update: If the user has specified this, all usage statistics are deleted. Otherwise, only the statistics for pages concerned by the object moves are deleted.

The authors show that DRO is computationally less expensive and yields better results on an OCB data base [DPS98] with 100,000 objects and 50 classes than the more complicated DSTC [BS96] algorithm.

Another interesting work has been published by McIver and King [MK94] who have presented an online clustering algorithm that takes into account the

OODB specific requirements when calculating object dependencies for reclustering. Two access patterns are considered for this purpose: references – when an object is recalled directly – and co-references – when an object is recalled due to its being referenced by another object that is currently being retrieved. Two measures are derived from these access patterns: The *heat* of an object is the frequency it has been recalled with in the past, i.e. the number of times it has been accessed. The *tension* of an object pair denotes the likelihood that the pair is accessed together. Speaking graph-theoretically, the heat has an interpretation as weight on the nodes, i.e. objects, while the tension denotes the edge weights.

When using greedy algorithms [BKKG88, HK89], the objects are clustered as follows: First, the object with the highest access frequency is chosen. Afterwards, depending on the algorithm, either a depth-first or a breadth-first structure is imposed on the nodes connected to this object, and objects are stored on the same page until the page is filled, at which point the object with the highest heat that has not yet been assigned is chosen as seed for the next page.

Both breadth- and depth-first traversal have their advantages, in accordance to the access patterns. McIver and King integrate both approaches into their algorithm and use the one better suited for the respective object. To this end, the algorithm must be able to make the distinction between set-oriented and navigational access statistics, so the heat measure is split to *set-heat* and *navigational-heat*. Accordingly, either a breadth-first or depth-first traversal is used on each node, depending on which access pattern has prevailed in the past.

For the reclustering, the task is split and distributed over three components: A statistics collector that keeps track of accesses, a cluster analyzer, and a reorganizer that is responsible for actually moving the objects according to the plan developed by the analyzer. The advantage of this architecture is that the first two do not require data base locks as their data structures reside in memory and that even when the cluster analyzer has run, the reorganizer only is triggered if the utility of doing so is above a certain threshold.

Although the process of reclustering is computationally expensive, the system shows that an intelligent architecture can compensate for such disadvantages by using offline computations that enable the component actually accessing the data structure to achieve a fast online reorganization. All three required operations are supported, but the cluster quality may be suboptimal for the greedy approach.

Summary

Obviously, the case of changing similarities is much less common in (centralized) applications than that of growing data sets. The first proposed algorithm, incremental k-means displays the well-known problems of its static counterpart: It is optimized for the detection of compact, spherical clusters, and the number of

clusters must be fixed in advance, which is especially critical with dynamic data sets.

Clustering in OODBs, on the other hand, has the disadvantage that the term cluster has a very particular interpretation in this context. It is used for describing the unit of data that fits into one page of external memory or can be transfered in one move from the external storage to the main memory.

As a summary, it must be stated that both method families are only of limited use in our scenario.

2.4.3 Mobile Scenarios

In the context of mobile, dynamic wireless, and/or ad hoc networks an important question is that of routing data between nodes that are not directly connected. For this purpose, information about the network structure is required. In contrast to conventional networks, there might not even exist designated routers; as a consequence, the routing tables of each participating node tend to be larger than in the conventional setting. As a countermeasure, the complexity of the network can be reduced by clustering the nodes and for example representing densely connected regions by their respective cluster centers. The cluster algorithms must work quickly and with as little communication overhead as possible; as we will see in this section, it is often considered more important that the clustering be stable than accurate. This last point implies that complete reclusterings of the data set are not desirable – even if the cluster structure could be improved – as long as the operation on the suboptimal structure is expected to impose smaller costs than the reclustering and the ensuing communication. All algorithms support the three required operations.

Krishna et al. [KVCP97] present an approach for clustering in mobile scenarios based on overlapping clusters in a cluster-connected graph. The graph $G = (V, E)$ is constructed over the set of mobile nodes V. E contains an edge for each pair of nodes that are connected. This presupposes that links are symmetric, which is not always the case, for instance when devices with different transmit and receive characteristics are used. The authors define k-clusters as clusters whose members can reach each other using at most k steps or $k - 1$ intermediate nodes. In order to obtain a clustering, overlapping 1-clusters, i.e. cliques in G are sought. If these clusters cover the whole graph and – in the case of a connected graph – each node in the graph can be reached from any other by a set of edges each of which only connect nodes in the same cluster, the graph is cluster-connected.

There are four basic cases that can take place in the network:

1. A node is switched on: In this case, the node asks its neighbors for their cluster tables and calculates a new set of 1-clusters that includes it in or-

der to ensure the reachability of the remaining graph. The authors show that at least one new cluster is thus formed. In order to avoid redundancy and, consequently, communication overhead, clusters that can be removed without disrupting cluster-connectedness are removed. This new set is then propagated.

2. A node is switched off: The neighboring nodes expand their clusters by searching for and including further nodes into the clusters. In this step, too, it is possible that new clusters are formed and/or old ones are removed.

3. A node connects to a new neighbor: This case can be reduced to case 1.

4. A node disconnects from one of its neighbors: This case can be reduced to case 2.

It is important to note that a rule must exist to decide which node executes the expansion of the clusters, but this problem can easily be solved for example by designating the node with the lowest or highest ID. Such a designation based on the ID has the advantage that it is easy to calculate and has low communication overhead. On the other hand, the authors note that due to their formulation of the cluster construction process, the cliques found by the joining nodes are not necessarily maximal.

Quite a few solutions rely on a hierarchical clustering of the mobile network, where usually so-called clusterheads coordinate the membership inside their respective clusters based on local visibility or connectedness. The role of the clusterhead is not fixed a priori but is assumed by members selected for instance by ID or connectivity. A general framework for clustering algorithms on the basis of this principle for different application classes can be found in the paper by Richa et al. [ROS01].

A rather simple cluster model serving for the coordination of resources and traffic inside cells or clusters has been put forward by Gerla and Tsai [GT95]. Their approach uses local clusterheads that coordinate the allocation in their respective cluster that is part of a wireless network. Two possible methods for finding a clusterhead are presented: First, the node with the lowest ID could be made clusterhead. Each node broadcasts its ID. If a node does not receive a lower ID than its own, it is a clusterhead. Gateways between clusters are in this case nodes that are connected to more than one clusterhead. The other possible scheme for obtaining a clusterhead is to locally elect that node among a set of nodes not yet connected to a clusterhead that has the highest connectivity – an approach that has similarities to the star clusters proposed by Aslam et al. [APR98] in section 2.4.1. Using a simulation, the authors show that the first approach provides a higher stability of the cluster structure. The support for cluster mobility is only sketched

and, according to the authors, follows from the cluster construction. The problem this approach entails, namely high load for the clusterhead and thus the existence of a possible bottleneck in the cluster, has motivated Lin and Gerla [LG97] to develop a more distributed structure. However, the actual determination of the cluster assignment by lowest ID from [GT95] is used. Clusters are defined by locality: There must exist a path with a length of no more than two hops between each pair of nodes in the cluster. If this constraint is no longer fulfilled due to the movement of a node, the node with the highest connectivity together with its neighbors is kept in the cluster. Nodes outside the cluster must determine a new cluster they can join.

Gao et al. [GGH$^+$01, GGH$^+$03] use the highest ID to identify a clusterhead. They model the mobile network using kinetic data structures [AABV99, FPT81], enclosing each node in a d-dimensional cube. If two cubes start or stop overlapping, the appropriate procedures for inclusion or exclusion are executed analogously to the other approaches in this section. The authors were able to show that their algorithm delivers a $O(1)$ approximation of the optimal discrete center problem that consists of finding a minimal set of nodes – the mobile centers – in such way that all nodes in the network are visible from at least one of the mobile centers.

A similar clustering algorithm has also been presented by Basagni [Bas99] based on the connectivity of the nodes as represented by the edge weights. He also treats the four basic network events listed above, additionally, if a node receives the clusterhead announcement from another node, a check is executed whether to join this new cluster or not.

Many protocols in the domain of mobile/ad hoc routing are reactive and only use the static information available during an important event like the removal of a cluster member. McDonald and Znati [MZ99, MZ02] have developed a more dynamic approach. Their approach includes an on demand, reactive routing strategy for inter-cluster routing and a probabilistic proactive intra-cluster routing component that takes into account the mobility of nodes. This separation is sensible since the intra-cluster mobility is usually much higher than the inter-cluster mobility, thus justifying the higher overhead of proactive routing. The clusters are marked by the mutual (α, t)-availability of all their members ((α, t)-clusters). Two nodes are said to be (α, t)-available if the probability is at least α that there exists a path between them after t units of time have elapsed, given that a path exists at the current time. This probability is estimated using a random-walk model for the movement of each node. In addition to reacting to the usual events in a mobile network as given above, each node uses an α timer that keeps track of the remaining life time of an (α, t)-cluster. When this time has elapsed, the node starts a proactive discovery to verify whether the (α, t)-availability of all other cluster members is still guaranteed. If this is not the case, the node leaves the cluster and

attempts to join another one.

Another approach by Har-Peled [HP04] might also be applicable in a mobile scenario if the movement is known a priori. The idea is to generate clusterings not only for the current time, but also for the future, given the predicted movement of the points. In the same vein, the algorithm by Li et al. [LHY04] maintains micro-clusters with a given cluster feature (analogously to BIRCH [ZRL96]) for sets of linearly moving objects.

Summary

The algorithms reviewed in this section deal with moving objects in the widest sense. One cluster of research is that of mobile/ad hoc networks, where the main objective is to facilitate communication between mobile devices by organizing the routing in accordance with the – constantly changing – topology of the network. Usually, the nodes that can easily reach each other are organized in clusters and designated nodes maintain the connection to other clusters.

The first problem when applying this class of algorithms to our scenario is that they use a crude binary "distance" measure – a node can reach another node directly – that cannot be easily generalized to more differentiated distance measures. The second general problem is that their focus does not lie – as we have seen – on a high quality of the resulting clusters but rather on low overhead and a fast reaction to topology changes, in most of the cases using heuristics. Taken together, the algorithms in this section are not sensibly applicable for our scenario where high-quality clusters based on differentiated similarities are required.

2.5 Randomized Algorithms

In the most general formulation given by Motwani and Raghavan, a randomized algorithm is *an algorithm that makes random choices during execution* [MR00]. Randomized algorithms are often used in cases where deterministic algorithms are too slow, i.e. have too high a computational complexity to solve a given problem in acceptable time. Furthermore, Motwani and Raghavan argue that randomized algorithms are often easier to describe and implement than their deterministic counterparts.

Two kinds of randomized algorithms exist: Las Vegas and Monte Carlo. The former class of algorithm always returns the correct solution, but its execution time varies depending on the characteristics of the data set and the random choices made by the algorithm. A Las Vegas algorithm is called efficient if its expected running time is bounded by a polynomial depending on the input size.

Monte Carlo algorithms, on the other hand, have a fixed running time, but may produce wrong or suboptimal results with a certain (small) probability. This may be acceptable for several reasons. First, if the problem cannot be solved in acceptable time by a deterministic algorithm, it is preferable to have a suboptimal solution instead of none. Second, the error probability can be made arbitrarily small by repeated executions of the Monte Carlo algorithm. And finally, Motwani and Raghavan show that any Monte Carlo algorithm can be transformed into a Las Vegas algorithm if it is possible to efficiently check the solution for correctness. In that case, it is sufficient to repeatedly execute the Monte Carlo algorithm until the result is correct. Efficiency for the Monte Carlo algorithm implies that its running time is polynomially bounded by the input size for the worst case.

2.5.1 Randomized k-Clustering

Inaba et al. [IKI94] consider the k-clustering problem with a stochastic sampling component. The k-clustering problem is defined as the problem of finding a partition of the data set into k nonempty sets that is optimal with respect to a given intra- and an inter-cluster criterion. In this case, the variance of the cluster members is used as intra-cluster criterion; the final k-clustering is obtained by recursively constructing 2-clusterings of the object set. As motivation, the authors mention the color quantization problem where colored pixels on a graphic display must be clustered according to their colors in order to be displayed using a given number of colors. A common problem at the time was to transform thousands of colors represented in the three-dimensional RGB space to the 256 colors of the VGA palette. A property of this problem is that the clusters are linearly separable [WWP88], i.e. two clusters located in their respective space can be separated by a hyperplane.

Using the centroid

$$\bar{x}(S) = \frac{1}{|S|} \sum_{x_i \in S} x_i \tag{2.23}$$

for the cluster S containing points x_i, the variance is given as the average squared distance between the cluster members and its centroid

$$\text{Var}(S) = \frac{1}{|S|} \sum_{x_i \in S} ||x_i - \bar{x}(S)||^2 \tag{2.24}$$

where $|| \cdot ||$ is a vector norm. Based on this, a parametrized variance

$$\text{Var}^\alpha(S) = |S|^\alpha \text{Var}(S) \tag{2.25}$$

with parameter $\alpha \in I\!N_0$ is defined. The authors claim that for higher values of α the cluster size becomes more homogeneous. The overall quality of the clustering is obtained by summing the variances $\text{Var}^\alpha(S)$ of the clusters. Assuming

that the number of clusters k is a constant, the problem is polynomially bounded, but nonetheless computationally quite expensive. With k not fixed, the problem becomes NP-complete.

The approach chosen by Inaba et al. is to estimate the variances by stochastically sampling m points of a cluster into a set T. For this set, all linearly separable 2-clusterings (T_1, T_2) are calculated along with their centroids t_1 and t_2. Since these 2-clusterings may not be optimal in terms of variance, the objects are reassigned to the two centroids by using the perpendicular bisector of the line segment between t_1 and t_2. This hyperplane generates another 2-clustering (S_1, S_2) with minimal variance given the centroids t_1 and t_2. For all these clusterings, the variances are computed and the clustering with the minimal sum of variances is kept. The authors show that it is admissible to use the sample sets to estimate clusters for the original sets since the expected values are identical.

In a later publication [IIK96] the authors show the application of the randomized algorithm to obtain initial clusterings for the k-means algorithm as well as some simulation results for varying sample sizes. The simulations indicate that small sample sizes are sufficient for obtaining good results that in most cases exceed those of the algorithm by Wan et al. [WWP88], especially if the separating hyperplane is not perpendicular to one of the axes.

A similar idea has been brought forward by Sabharwal and Sen [SS05]. Instead of fixing both centroids, they fix only one point as approximate centroid and use a set of lines passing through this point. The lines cover only a predefined angle. The idea is to project the other points on each of these lines and to find the centroid of the second cluster that minimizes the cost of the clustering by random sampling.

2.5.2 Evolutionary Clustering

Gorunescu and Dumitrescu [GD03] present an approach to clustering that relies on evolutionary principles [Hol75]. Based on the ideas of incremental clustering presented in section 2.4.1, they define the evolutionary operators as follows:

- Each population member is a possible clustering. Its chromosome contains a gene for each object to be clustered, the gene containing the index number of the cluster the object is associated with. Zero means that this object is not (yet) assigned. For the initial population, the algorithm stochastically selects one gene in each chromosome and sets it to one, the other genes have zero values. As a consequence, the members of the population most probably cover a large area of the object set because the initial clusters are evenly distributed among the objects.

- The performance of a chromosome is evaluated by summing the pair-wise distances of the members of each cluster, where the distance between two objects is a weighted Euclidean distance measure. This value is divided by the number of clusters in order to prevent singleton clusters.

- The recombination is done by a one-point crossover operator that may either split or merge clusters. The crossover operator cuts each parent's gene string in two parts – both at the same place – and concatenates the first part of the first parent's gene string with the second part of the other and vice versa. Of the two parents and two children, the two fittest specimens are kept in the following selection phase.

- Two mutation operators are employed: The first splits clusters by assigning an object to a new cluster if it is not alone in its old cluster. The other one moves an object from one cluster to the other.

- In order to maintain the incremental nature of the algorithm, an incremental operator is used that assigns an object whose corresponding gene has the value zero to either a new or an existing cluster.

- The algorithm stops when no progress is made over the last iterations.

The authors offer a small evaluation on a fictional weather data set. The clusters that the authors deem relevant are found in nine cases out of ten.

Furthermore, the algorithm by Gupta and Grossman [GG04] introduced in section 2.4.1 is an evolutionary algorithm for incremental or stream data.

2.5.3 Clustering with Simulated Annealing

Simulated annealing [KGV83] is an optimization technique that mimics the behavior of cooling matter from a heated state where its molecules can freely move to a frozen or solid state where the structure is fixed. It is based on the Metropolis Monte Carlo algorithm developed by Metropolis et al. [MRR$^+$53] where this process was first modeled using a computer.

The principle is that of a relaxed hillclimbing algorithm: An initial solution is iteratively altered by exchanging or permuting some of its components. These new solutions are then evaluated. If such a solution is better than the previous one, it is accepted as input for the next iteration. If it is not better, the probability of its being accepted depends on the current temperature and the cost difference to the current solution. The general formula for the acceptance probability is

$$p(\text{accept}) = e^{\frac{-D}{K_b T_m}} \tag{2.26}$$

where D is the cost difference, T_m is the current temperature and K_b is the Boltzmann constant. For models where the temperature variable does not correspond to any real temperature, the acceptance probability can be simplified to

$$p(\text{accept}) = e^{\frac{-D}{T_m}} \tag{2.27}$$

The temperature-dependent acceptance rate allows the algorithm to easily overcome local minima at the beginning. Towards the end, the global exploration by the algorithm becomes more and more local, because only smaller and smaller deteriorations of the current solution have a sufficiently high probability to escape a local minimum.

The general schema of a simulated annealing algorithm has been given by Mitra et al. [MRSV86]:

```
X := j_0
m := 0
while (termination criterion not met)
begin
   while (inner loop criterion not met)
   begin
      j :=GENERATE(X)
      if (ACCEPT(D, T_m))
         THEN X := j
   end
   T_{m+1}=UPDATE (T_m)
   m = m + 1
end
```

GENERATE creates a new solution, ACCEPT decides whether the solution, having a cost difference of D, should be accepted at temperature T_m. UPDATE is responsible for lowering the temperature according to the annealing schedule.

The inner loop, where the temperature is kept constant can also be interpreted as a general Markov chain. It usually ends when a predefined number of moves has been executed. The outer loop normally terminates at a given temperature – the frozen state – or when no change in the objective function could be observed in the last rounds. For the annealing schedule $T_{m+1} = 0.9T_m$ it has been shown that the optimal solution is found when the number of inner loop iterations at each temperature level is infinite [MRSV86]. A more general assertion for the minimization problem has been given by Hajek [Haj88]: He defines cups as the (reachable) neighborhood of local minima; the depth of a cup is given as the difference of the maximal value outside the cup and the local minimum inside. If

the annealing schedule $T_k = \frac{c}{\log(k+1)}$ is used, where c is a constant greater or equal to the depth of the deepest cup, the probability of finding the global minimum is 1.

In 1990, Bell et al. [BMSM90] presented different approaches for clustering objects in (relational) data bases in such a way as to minimize access time for predefined queries. They distinguished two layers for the data organisation: Tuples are organized in pages, and these in turn are grouped in cylinders. Access times for objects on the same page are minimal while access for objects on different cylinders is slowest. Their first approach, having run time $O(n)$ according to the authors borrows concepts from single linkage clustering and shall not be detailed here.

The initial configuration for the simulated annealing algorithm assigns each data tuple to its own page. The initial temperature is set in a way as to accept almost all negative changes. The GENERATE function selects a tuple and a page to which this tuple should be moved. ACCEPT follows the general scheme given above, UPDATE is given as $T_{m+1} = 0.9T_m$. For the inner loop criterion, a threshold for the number of changes as well as for the number of changes per tuple is used. If the latter threshold is met, the number of failures F is increased. If $F = 3$ or $T_m = 0.01$, the outer loop terminates.

The results show that the algorithm performs better than the other algorithms known at that time if there is a high connectivity between elements from different relations (high fanout).

Hua et al. [HLL94] have considered the problem of using simulated annealing on large data sets where the similarity graph cannot be fitted into the main memory of a computer. Due to the stochastic nature of the algorithm, conventional buffer replacement strategies have a very low performance and may result in page thrashing, thus reducing throughput to that of the disk subsystem.

For this reason, the authors propose a decomposition of the similarity graph in order to sequentially cluster parts of the graph. T_0 is set in such a way that 95% of all negative changes are accepted, UPDATE is again $T_{m+1} = 0.9T_m$, the inner loop terminates after βn iterations, with $\beta \in [5, 8]$ and the outer loop terminates when the temperature drops to 0.001 and the cost function is unchanged over two iterations. The difference to [BMSM90] is that the algorithm starts with v randomly chosen subgraphs. Of these, k are considered simultaneously. In each step of the algorithm, either a node-perturbing move – swapping the association of two nodes to their respective subgraph – or a buffer-perturbing move – exchanging the k subgraphs by k other randomly chosen ones – is executed according to the externally given probability parameter c for the buffer-perturbing move.

As a result, the number of disk I/Os could be drastically reduced, especially if the size of the buffer is small in relation to the total data base size.

2.5.4 Random Walk Circuit Clustering

Motivated by the growing complexity in the field of integrated circuit (IC) design, Cong et al. [CHK91, HK92] have studied the usability of random walks for the partitioning of circuit graphs. Each IC can be represented as a netlist graph, where the nodes symbolize the components and the edges mirror their electrical contacts. In order to minimize run time and resistance, and thus heat generation, it is imperative for the design of an IC to group closely collaborating groups of elements in direct neighborhood on the die. Traditionally, top-down partitioning is employed for the task of grouping the elements on an IC. Unfortunately, these techniques are usually NP-complete and therefore demand more and more computational power with growing IC size. In order to reduce complexity, the authors propose the use of a cluster algorithm as a preprocessing step. They present a method that consists of finding clusters by identifying cycles in a random walk on the IC's graph representation. These clusters can then be replaced by single nodes that inherit all edges of their cluster members and that are used as input for a traditional top-down partitioning.

As a first step, the authors develop what they call a "proper clustering metric", the degree/separation (DS) metric. The degree of a cluster is the average number of nodes that are incident to a member node of that cluster and have at least two neighbors in the cluster. The separation is defined as the average length of the shortest paths between all pairs of cluster members. The authors claim that the DS measure is a robust measure for cluster quality.

In order to detect clusters, a random walk is executed on the netlist graph until all nodes have been visited at least once. This is called the cover time and is defined as the maximum expected length of a walk that visits all nodes over all possible starting nodes. According to Kahn et al. [KLNS89], the cover time is between $O(n^2)$ and $O(n^3)$, depending on the graph's properties like form, average degree etc. During the running time, cycles begin to emerge in the walk. A cycle in this context is defined as a node sequence of the form $\{v_p, \ldots, v_q\}$ with $v_p = v_q$ and $v_j \neq v_i, i, j \in \{p, \ldots, q-1\}$. The authors state that the members of such a cycle form a cluster, since per construction the set of objects contained in the cycle is as tightly coupled as possible: If there were a cycle containing less, more tightly connected elements, it would be found by the random walk with high probability.

After a sufficient number of steps has been taken by the walk, the longest cycle $C(v_j)$ is computed for each node. The binary relation $v_a \bowtie v_b$ is defined to be true if $v_a \in C(v_b)$ and $v_b \in C(v_a)$. Clusters are generated using the transitive closure of the $\bowtie$ relation.

In a later paper [HK92], the authors enhance the method by introducing the sameness of nodes. The sameness of node u to node v is calculated according to the similarity of the cycles starting from them.

In tests, the algorithm performed better than the conventional algorithms used for this domain; this was especially evident when the algorithm was used for generating an input clustering for the Fiduccia-Mattheyses heuristics [FM82], an algorithm that is frequently used for top-down partitioning of ICs.

2.5.5 Randomized Generation of Hierarchical Clusters in Ad Hoc Networks

A two-stage model using first a randomized, then a deterministic algorithm for the construction of star-shaped clusters in wireless networks, each consisting of a master and several slaves has been developed by Ramachandran [RKSA00]. In the first phase, a repeated Bernoulli trial is executed by each node, typically with a small probability p of success. If the node succeeds in at least one of these trials, it is marked as a master-designate, else it is a slave-designate. By collecting responses from slave-designates, the master-designate may become a master. One of the masters is elected – either after reaching a timeout or after receiving a sufficient number of responses from other clusters – as super-master that deterministically corrects possible errors in the star cluster structure.

2.5.6 High-Speed Noun Clustering by Hash Functions

Ravichandran et al. [RPH05] have presented a randomized algorithm to quickly compute the cosine similarity [Sal88] (cf. Eq. (1.12)) of objects in the domain of Natural Language Processing (NLP). With n objects in a k-dimensional feature space, the running time for the deterministic computation of the cosine similarity matrix is $O(n^2k)$. The algorithm detailed here is interesting as a preparatory step for cluster algorithms using similarity measures rather than feature vectors as input.

The first aspect of the authors' solution is to use Locality Sensitive Hash (LSH) functions [Cha02]. Contrarily to conventional cryptographic hash functions where small differences between the string or binary representation of an object should entail large differences in the hash value, the LSH achieves quite the opposite: Similar objects have a high probability of obtaining the same hash value.

First d additional random vectors $\{r_1, \ldots, r_d\}$ with k dimensions and of unit length are generated; d should be considerably smaller than k. The hash function

$$h_r(u) = \begin{cases} 1 & \text{if } r \cdot u \geq 0 \\ 0 & \text{if } r \cdot u < 0 \end{cases} \tag{2.28}$$

using the vector scalar product is applied to the d vectors for each vector u that represents an object. The probability that the random hyperplane generated by

r separates two vectors u and v is proportional to the angle Θ between the two vectors; for the proof refer to Goemanns and Williamson [GW95]:

$$P(h_r(u) = h_r(v)) = 1 - \frac{\Theta(u,v)}{\pi} \tag{2.29}$$

In other words, the cosines of the two vectors can be written as

$$\cos(u,v) = \cos((1 - P(h_r(u) = h_r(v)))\pi) \tag{2.30}$$

By sampling d vectors, where d is highly domain dependent, an estimator for the cosine similarity is obtained. Furthermore,

$$P(h_r(u) = h_r(v)) = 1 - \frac{H(u,v)}{d} \tag{2.31}$$

with $H(u,v)$ the Hamming distance or the number of differing bits between u and v. Thus, an algorithm is able to find all pairs with a cosine larger than a given threshold value by repeatedly permuting and sorting the bit streams or signatures $\bar{u} = \{h_{r_1}(u), \ldots, h_{r_d}(u)\}$ and checking each object's closest neighbors for their Hamming distance. Unfortunately, Ravichandran et al. do not give any information about the convergence of the procedure, their only claim is that the number of "random vectors is highly domain dependent".

The algorithm is likely to find similar pairs in at least one of the permutations, since a high similarity between two objects implies a low Hamming distance between the bit stream representations and thus a high probability that the two are within distance B in at least one random permutation. Using a parameter $B = 100$ and a data set of 6GB (TREC9 and TREC2002), an accuracy of more than 80% for the top 10 lists could be achieved using 1000 permutations of the bit stream representations.

2.6 Random Walk Theory and Markov Processes for Clustering

The approaches presented in this section profit from the theory behind random walks and especially Markov processes. They are however not randomized, but deterministic and rely in general on some sort of transformation of the transition matrix.

2.6.1 Clustering by Markovian Relaxation

The algorithm developed by Tishby and Slonim [TS01] is based on deriving a Markov process from the distance matrix of an object set and afterwards relaxing this process.

In order to transform the distance matrix into the transition matrix of a Markov process, the objects are interpreted as the states of the chain. Since distances along a path in a graph are additive, but probabilities are multiplicative, the transition probabilities are constructed by exponentiating the distances:

$$p(x_i(t+1)|x_j(t)) \propto e^{-\lambda d(x_i,x_j)} \tag{2.32}$$

where λ is a scaling factor in order to normalize the probabilities. The result of this transformation is the stochastic matrix P.

$$p(x_i(t)|x_j(0)) = P_{ij}^t \tag{2.33}$$

denotes the probability of visiting state x_i at time t after starting in state x_j at $t = 0$. As $t \to \infty$, $p(x_i(t)|x_j(0))$ approaches the stationary distribution π. By increasing the parameter t the influence of the starting point on the probability distribution of the current state is relaxed, hence the name Markovian relaxation. In other words: For large values of t the information about the starting point is gradually lost. This information loss is measured and used as a criterion for suitable clusterings. The proposed information measure is the mutual information

$$I(X,Y) = \sum_{x\in X, y\in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} = \sum_{x\in X, y\in Y} p(x)p(y|x) \log \frac{p(y|x)}{p(y)} \tag{2.34}$$

between two random variables: In this case, the starting state X_0 and the current state X_t are used. Thus, as an information measure for the Markovian relaxation at time t, we obtain

$$I(t) = I(X_0, X_t) = \sum_j p_j \sum_i P_{i,j}^t \log \frac{P_{i,j}^t}{p_i^t} \tag{2.35}$$

with p_j the prior probability of j and $p_i^t = \sum_j p_{i,j}^t p_j$ the unconditional probability of visiting state i at time t. Note that the second sum in Eq. (2.35) is the Kulback-Leibler divergence, a measure for the similarity of distributions [CT91]. The dissimilarity approaches zero as $t \to \infty$. Analyzing the first derivative of $I(t)$, regions in the parameter space can be isolated where little variation is observed, i.e. where the transition matrix for t and $t+1$ is nearly unchanged. In that case, objects with very similar rows in P^t form a cluster.

A similar approach was also presented by Szummer and Jaakkola [SJ01] for labeling big sets of points using few labeled examples. Instead of including all distances in the computation of the transition matrix, they only included the K nearest neighbors of each object.

Another stochastic approach to clustering by Gdalyahu et al. [GWW99] is based on transforming the similarity matrix of the object graph and subsequently

thresholding it. The components of the thresholded graph are then interpreted as clusters.

For the transformation of the similarity matrix, the authors use r-way cuts. An r-way cut is a partitioning of the graph into r disjoint parts; the capacity of the cut is the sum of the edge weights that connect members of different parts. Contrary to min-cut algorithms [WL93] not only the minimal cut, but all possible cuts are considered for this transformation. Each cut is assigned an occurrence probability that decreases with increasing capacity of the cut. For each level r, i.e. number of clusters, the probability p_{ij}^r is computed that the nodes i and j are in the same partition in an arbitrary r-cut. Unfortunately, the complexity of calculating all cuts is rather high, but as Karger and Stein have shown [KS96], the cut probability decays fast with rising capacity such that the p_{ij}^r can be bounded using only low-capacity cuts. As a consequence, it is sufficient to generate a polynomially sized sample of cuts that are generated according to their probabilities and to estimate the p_{ij}^r based on the sample.

The authors show that the method is capable to cleanly separate three intertwined spiral clusters in the presence of noise and avoids the bridging effect that could be expected by clustering the data set using single linkage clustering.

2.6.2 Text classification with Random Walks

Xu et al. [XYZ06] propose to use absorption probabilities of random walks to classify texts. They assume a small set of labeled samples from which the labels of the unlabeled data are derived. For this purpose, the similarity matrix $W = (w_{ij})$ is once again transformed to transition probabilities of the stochastic process

$$p_{ij} = \frac{w_{ij}}{\sum_j w_{ij}} \tag{2.36}$$

for the unlabeled data. The labeled objects, on the contrary, are represented by absorbing states of the Markov chain, i.e. $p_{ii} = 1$. The correct label for an unlabeled sample j can then be found by identifying that absorbing state that has the highest probability for absorbing a walk starting at j.

The authors also present an approach for including new data into the classification. However, in the context of this work, this algorithm is not applicable since it is based on the assumption that there can be no transitions from the old to the new data, i.e. the arrival of new data does not disturb the old classification.

2.6.3 Clustering by Flow Simulation

Van Dongen [vD00a, vD00b, vD00c] has proposed a method called graph clustering by flow simulation or Markov Cluster algorithm (MCL) that is especially

suited for sparse graphs. Interpreting the pairwise similarities between nodes as a flow, the general idea is to reinforce the flow inside the clusters and to weaken flow between clusters by alternately applying expansion and inflation operators to a stochastic matrix of transition probabilities, until the inter-cluster flow disappears or, in van Dongen's words, "dries up".

When considering random walks on a graph, the author states that a walk of length k started from a node inside a densely connected region, i.e. a cluster, has a high probability of ending in the same region, at least for small values of k. The expansion and inflation operators serve to amplify this effect.

The input graph is used for the construction of a flow matrix for the underlying process by row-wise normalization of the entries in the graph's similarity matrix. The first operator, expansion, is the k-th power of the flow matrix that, as seen in section 2.1.4, returns the probability of a k-step transition between two nodes of the graph. In this way, it enables the node to potentially discover new neighbors that are only indirectly connected to it. Asymptotically, the k-th power of the transition matrix returns the stationary distribution; it is thus important to choose the correct parameter k. The expansion serves to identify links inside the cluster. However, the distinction between edges inside the clusters and those between two clusters is not very pronounced after this step.

Therefore, the inflation operator is introduced as the Hadamard-Schur product of the flow matrix with itself. The Hadamard-Schur product for two matrices, $A \circ B$ has the entries $[A \circ B]_{ij} = a_{ij}b_{ij}$. In order to restore the stochastic nature of the matrix, the result of this computation is afterward multiplied with a diagonal matrix using standard matrix multiplication. In sum, the Hadamard-Schur operator serves to enhance the distinction between intra- and inter-cluster edges by strengthening the former and weakening the latter.

Although the method has a considerable complexity – it converges quadratically in the number of matrix multiplications– the author claims that it has been used with success on large data sets incorporating different cluster shapes.

2.6.4 Separation by Neighborhood Similarity and Circular Escape

The idea of modifying object similarities in order to sharpen the distinction between inter- and intra-cluster edges has also been pursued by Harel and Koren [HK01]. The algorithm expects a graph $G = (V, E, \omega)$ as input. The authors claim that by iteratively applying one of the two separation operators detailed below, it is possible to let the similarity matrix converge to a binary matrix where intra-cluster edges are weighted with ones and inter-cluster edges, the so-called separators, receive zero weights. The first of the separation operators is the separation

by neighborhood similarity based on the same idea as van Dongen's expansion operator. For this measure, the vector $P^k_{visit}(i)$ denotes the probability distribution of a random walk starting at node i after k steps, i.e. the j-th component is the probability of visiting node j in this step. Moreover, $P^{\leq k}_{visit}(v) = \sum_{i=1}^{k} P^i_{visit}(v)$ aggregates the probabilities for all steps up to k. The role of k is to select the degree of locality: For $k = 1$ only the one-step distribution over the node's neighbors is consulted, whereas for $k \to \infty$, P^k_{visit} converges to the stationary distribution that, as we have seen in section 2.1.4, does not depend on the starting node. As a result, the method produces a graph $NS(G)$ with weights $\omega_s(u, v) = sim^k(P^{\leq k}_{visit}(v), P^{\leq k}_{visit}(u))$ where sim^k is some measure for the similarity of vectors.

Separation by circular escape is based on contemplating the probabilities $P_{escape}(v, u)$ that denote the probability that a random walk starting at v first visits u before returning to v. The resulting graph $CE(G)$ uses the weights $w(u, v) = P_{escape(u)}(v, u)P_{escape}(u, v)$. Both separations can be computed in $\Theta(|E|)$ for similarity graphs of bounded degree.

For the clustering, it is either possible to use the new graph $NS(G)$ or $CE(G)$ as input graph for another cluster algorithm, or to directly cluster based on the matrix of weights. To this end, the authors introduce a threshold value that separates inter- and intra-cluster edges. Clusters can then be found by identifying connected subgraphs in the graph containing only the intra-cluster edges. Using different synthetic data sets for evaluation, the authors show that the method performs well in separating clusters, even if they overlap.

2.7 Summary

This chapter has built the foundation for the main part of the thesis, both by introducing basic concepts like random processes and randomized algorithms and by reviewing the state of the art in dynamic clustering. As the discussion in the respective sections has shown, no algorithms exists that are able to fulfill the requirements for our scenario. Either, the algorithms cannot cope with changing similarities like the methods in section 2.4.1, have differing cluster concepts like the OODB and mobile approaches, or produce clusters of suboptimal quality due to computational or bandwidth restrictions.

This lack of suited algorithms has lead to the development of the restricted random walk update algorithm that is at the core of this thesis.

Chapter 3

Restricted Random Walk Clustering

Since – according to the literature overview of the last chapter - no existing algorithm fulfills all the requirements stipulated in chapter 1 for cluster updates, a new cluster update approach will be presented here that is based on restricted random walk (RRW) clustering. The presentation and discussion of the update algorithm in chapter 4 is preceded by the introduction of the basic RRW cluster algorithm in this chapter. The algorithm is composed of two stages, the walk and the cluster construction stage; these will be presented in the first section. In the second section some of the method's properties are discussed; the chapter ends with an overview of the applications RRW clustering has been employed for together with the evaluations that have been performed in that context.

3.1 RRW Clustering

In this section, we will consider a specific kind of random walk, the restricted random walk as defined by Schöll and Schöll-Paschinger [SP02, Sch02, SSP03] and see how it can be used to construct clusters of different characteristics. The basic idea is to first execute random walks on the data set such that with growing length of the walk only more and more similar objects are selected. The cluster construction relies on the assumption that pairs of objects occurring at late stages of one of these walks have a higher tendency to belong to the same cluster than the pairs that occur only at the walks' beginning where random influences still dominate structure information.

Following the categories given in the introduction in chapter 1, the algorithm is a hierarchical, stochastic, and randomized algorithm that works both on dissimilarities and similarities either in metric spaces or on graphs. Depending on the cluster construction method, it produces disjunctive or overlapping clusters.

3.1.1 The Walks

The algorithm received its name from the restriction on the choice of successor states during the walks that enables them to detect clusters. If we consider an object set with pairwise distances between the objects, a walk starts at its starting node i_0 and then, with each transition or step taken, traverses smaller and smaller distances. Fig. 3.1 (adapted from [Sch02]) demonstrates this principle for a walk starting at node i_0: With each step, the circles containing possible successors become smaller and smaller until, for i_5, the circle is so small that it only contains the current node i_5 because the walk has arrived at i_5 via its nearest neighbor, i_4.

Transferred to a similarity graph $G = (V, E, \omega)$ with V the set of objects, E containing an edge for each pair of objects with a positive similarity, and ω_{ij} the pairwise similarity between two nodes i and j, this means that if the walk arrives at node j via the edge (i, j), only neighbors of j are considered that are more similar to it than i. This leads to higher and higher similarities between consecutive nodes. Finally, the walk ends when there are no more edges present that satisfy the RRW restriction.

Two alternative formulations of the walks exist. Let us start by considering the original one by Schöll and Schöll-Paschinger in which the walk process is a reducible, finite second order Markov chain. In order to facilitate the analysis

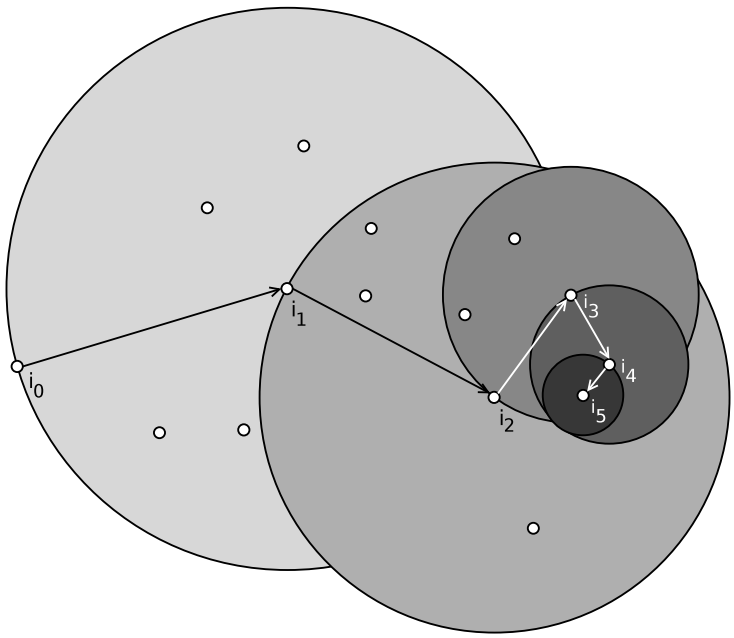


Figure 3.1: Converging circles during a restricted random walk with length five, starting at i_0

of the process, an alternative but asymptotically equivalent formulation as irreducible infinite first-order Markov chain was developed in [Fra03] that we will discuss afterwards. Another possible way to obtain Markov chains for the process is outlined in [Sch02].

In the original formulation, the objects to be clustered are contained in a set X and a pairwise distance measure $d(i, j)$ exists for all pairs (i, j) of objects in X. The alternative formulation operates on a graph representation where the set of objects and their similarities are given as a (finite) similarity graph $G = (V, E, \omega)$. Note that the original approach was formulated for distances, the alternative one on similarities. This, however, is merely a formality since distances and similarities can easily be transformed into each other (cf. e.g. [KR90]). As an alternative, the comparison operators $>$ and $<$ can be interchanged in order to make the algorithm run on the respective measure. Note though that there might be special cases like a zero similarity that translates to an infinite distance.

Consider first the original formulation of the walk process by Schöll and Schöll-Paschinger. Each walk starts at an arbitrary node i_0. In order to cover the whole object set, Schöll and Schöll-Paschinger suggested to start a fixed number of walks from each node.

The first successor, i_1, is chosen among all objects in the set X, and the distance between the two objects is recorded as the step width $s_1 = d(i_0, i_1)$ for the first step. For each following step the restriction is introduced that the distance between the current and the next node must be smaller than the one between the current node and its predecessor. Formally, if the step width is

$$s_m = d(i_{m-1}, i_m) \tag{3.1}$$

in the m-th step, this is achieved by selecting the $m + 1$st node from a uniform distribution over the members of

$$T_m = \{j \in X | d(i_m, j) < s_m\} \backslash \{i_m\} \tag{3.2}$$

The selection is repeated until T_m is empty, i.e. until no admissible successor exists. i_m is excluded since repeated visits to the same node in consecutive steps are of little use for the detection of clusters.

In this formulation of the method, the states of the stochastic process are defined as the objects in X and transitions take place according to the pairwise distances. This has the disadvantage that the transitions of the stochastic process depend on the two last steps; as a consequence the Markov condition for a first order Markov chain is violated.

In order to obtain such a chain, let us consider a slightly differing formulation of the process: The walks are defined on the edges of the graph, consequently a single walk has the form $(e_0, e_1, \ldots, e_k) = ((i_0 i_1), (i_1, i_2), \ldots, (i_k, i_{k+1}))$. As

we will see below, the two formulations are equivalent in terms of the probability distribution of the walk realizations.

Another problem is the choice of the first node. Although iterating through all nodes and starting the same number of walks from each of them guarantees a complete coverage of the object set, it may not be optimal since it does not take into account local particularities like the degree of a node or more generally the density of a graph region. For the following formulation, this selection process is given by a stochastic selection of the first node of the start edge e_0 from a uniform distribution over V.

Walks in the original formulation are finite. In order to obtain an infinite, irreducible chain, imagine the single walks being concatenated. As a delimiter between two walks we introduce the start/end state Ω that represents the transition between two consecutive walks and is also the initial state of the Markov chain. From there, a first edge $e_0 = (i_0, i_1)$ is chosen from the set of all edges with a probability of

$$P(e_0|\Omega) = P((i_0, i_1)|\Omega) = \frac{1}{|V| \deg(i_0)} \tag{3.3}$$

with $\deg(i_0)$ the degree of node i_0. From the point of view of the probability distribution over all nodes and edges, this is equivalent to the original formulation: $|V|$, the first part of the denominator represents the choice from a uniform distribution over all nodes; $\deg(i_0)$, the second part, stands for the uniformly distributed choice among the links starting from i_0.

For the choice of successor edges, the set of all admissible successors of an edge $e_m = (i_m, i_{m+1})$ is defined as:

$$T_{e_m} = T_{(i_m, i_{m+1})} = \{(i_{m+1}, j) \in E | \omega_{i_{m+1},j} > \omega_{i_m,i_{m+1}}\} \tag{3.4}$$

i.e. all edges incident to the target node of the current edge and having a higher weight than the current edge are considered as possible successors. The actual successor is determined by randomly selecting one member of T_{e_m} using a uniform distribution. The walk stops when T_{e_m} is empty. In that case, the Markov chain enters once again the state Ω and another walk is started.

As a consequence, the process is now given as a irreducible, infinite first-order Markov chain: It is irreducible since

1. Each state – i.e. edge – can be reached from Ω. This is guaranteed since there is a positive transition probability from Ω to every state as given by Eq. (3.3).

2. Ω, in turn, can be reached from each state. Were this not the case, infinite restricted random walks would be possible which, by construction, cannot

happen on a finite graph. Due to the finiteness of G and the strictly increasing similarity in each step, every walk must reach an edge that has a higher associated similarity than all of its neighbors, and thus the state Ω is entered.

As a consequence, each state can be reached from each other state, which means that the process is irreducible; furthermore, it is clearly infinite by construction.

The two formulations are equivalent for the purposes of clustering: The newly introduced transition state Ω is ignored when constructing the clusters, as will be seen in the next section. The selection probabilities of the first edge are modeled in such a way as to be equivalent to the behavior of Schöll and Schöll-Paschinger's algorithm for the case of a uniform choice of the first node from the set of all nodes. For the other case, where a constant number of walks is started from each node, the selection process for the transition from Ω must be redefined accordingly.

Finally, it should be noted that starting a fixed number of walks from each node and choosing the first node from a uniform distribution leads to the same expectation for the distribution of the walks in the asymptotic view, i.e. for an infinite number of walks. The order in which the walks are executed may vary, but we can expect that every node is chosen the same number of times as starting node when using the uniform distribution, i.e. the same number of times it would have been used when starting a fixed number of walks. Since the rest of the walks is defined identically and since the order in which the walks are executed is not considered for the cluster construction, the resulting clusters are expected to be identical.

For an example of the walk process, consider the section of an example graph shown in Fig. 3.2 – the dashed lines symbolize its connections to the remainder of the graph; they will be ignored for the example. The transition matrix for this graph section is given in Tab. 3.1.

Let us now consider an exemplary realization of the Markov process on this graph. The process starts in state Ω. From there, let us say that DF is chosen as e_0. The probability for this event is $\frac{1}{7*2} = \frac{1}{14}$ since V has seven elements, and selecting D as i_0, the algorithm has the choice between its two neighbors. We obtain $T_{e_0} = \{FB\}$, thus the choice of $e_1 = FB$ is clear. Arriving at B via the edge with weight 2, the edge BG is excluded due to its weight being to low. Consequently, $T_{e_1} = \{BA, BC\}$. Each of them has a probability of $\frac{1}{2}$ of being chosen, let us say that the process picks BC. From $T_{e_2} = \{CA, CG\}$, CG might be chosen as successor. At that point, $T_{e_3} = \emptyset$ and the process returns to Ω to continue, say, with the walks EGC, BGE, ACG, BCA, $CBACG$, $GBCA$, and $DACG$.

Table 3.1: transition matrix for the graph in Fig. 3.2

	Ω	AB	AC	AD	BA	BC	BF	BG	CA	CB	CG	DA	DF	EG	FB	FD	GB	GC	GE
Ω	0	$\frac{1}{21}$	$\frac{1}{21}$	$\frac{1}{21}$	$\frac{1}{28}$	$\frac{1}{28}$	$\frac{1}{28}$	$\frac{1}{28}$	$\frac{1}{21}$	$\frac{1}{21}$	$\frac{1}{21}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{7}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{21}$	$\frac{1}{21}$	$\frac{1}{21}$
AB	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AC	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
AD	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BA	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BC	0	0	0	0	0	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	0	0	0	0	0
BF	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$
CA	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CB	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CG	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DA	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
EG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
FB	0	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0
FD	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GB	0	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0
GC	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GE	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

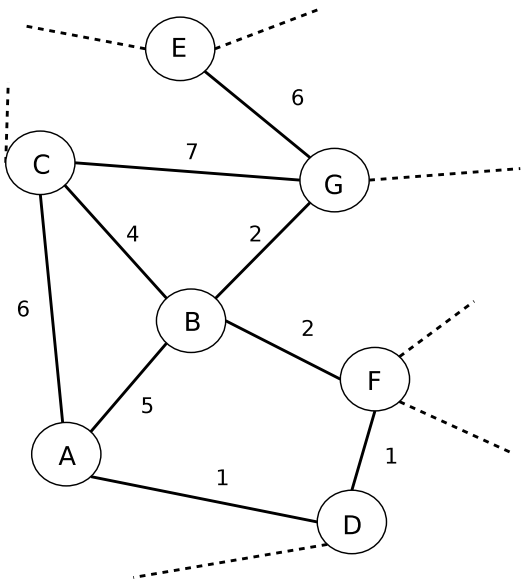


Figure 3.2: Section of an example similarity graph

3.1.2 Cluster Construction Methods

In the walk stage described in the last section, restricted random walks were used in order to explore the structure of the similarity graph. In a second stage, it is now necessary to make use of the information thus collected to construct clusters. The basic idea as sketched in the introduction to this chapter is that the later in a walk a pair of nodes occurs, the higher is their tendency to belong to the same cluster. In the following sections two cluster construction methods are introduced that implement this principle: component clusters and walk context clusters. These approaches return substantially different clusters: While the first produces very large, disjunctive clusters, the clusters resulting from the walk context are much smaller and overlapping. The question which one is better always depends on the application's requirements.

In both cases, the actual outcome is a hierarchy of clusters. In the case of disjunctive clusters, this hierarchy can be represented by a dendrogram like the one in Fig. 1.2. As discussed in the first chapter, a partitional clustering is obtained from the hierarchy by applying an horizontal cut through the dendrogram at a given cutoff level. The applications section contains a discussion of how to set this level.

Component Clusters

The original method for cluster construction put forward by Schöll and Schöll-Paschinger [SP02] is that of component clusters. A series of graphs $G_k = (V, E_k)$ is constructed for all applicable levels k such that E_k contains an edge between two nodes iff they have formed the k-th step of any walk. It follows that the connections in G_k are the more meaningful the higher k is. From this series, a second series of graphs

$$H_k = \cup_{l=k}^{\infty} G_l \tag{3.5}$$

is derived that contains edges for all node pairs occurring at level k or above in any walk. Clusters are then defined as components, i.e. connected subgraphs of H_k.

Returning to our example with the walks $DFBCG$, EGC, BGE, ACG, BCA, $CBACG$, $GBCA$, and $DACG$ we obtain the components shown in Tab. 3.2.

As can be seen, the cluster hierarchy is captured well in this example. At the lower levels two and three the weakly connected nodes (first D, then E and F) are separated from the central cluster. Finally, on the highest level, the "central" pair CG is correctly found.

Schöll and Schöll-Paschinger have evaluated RRW clustering against single linkage (SL) and complete linkage (CL) in [SSP03]. Their first evaluation data set

contains compact and elongated, overlapping clusters. Both CL and RRW outperform SL in this case, since the latter algorithm cannot discern overlapping clusters or clusters connected by bridges very well. The difference between CL and RRW in this case was visible in the treatment of outliers that were included into the clusters by the CL algorithm, but not by RRW. The second data set contains a ring shaped cluster that encloses a spherical one. Here, RRW is able to identify the two clusters correctly. SL and CL discovered four subgroups in the enclosing cluster and began merging them with the inner cluster. When the distance between the two clusters was increased, at least SL could find the correct clustering.

In spite of these promising results, there is still potential for improvements, especially in the application context given by e.g. library data. For example, the step level concept used here lacks some granularity: The nodes E and F are removed at the same level in spite of the strongly differing similarities of the edges linking them to the central cluster. A further disadvantage of the component clusters turned out to be their size: The clusters for the library data set described in section 3.3 proved to be very large – sometimes hundreds of thousand of documents; the average cluster size was 27,000. Although Schöll and Schöll-Paschinger state that the bridging effect, i.e. the unwanted merging of small clusters into one large cluster, is noticeably smaller than with single linkage clustering, it could nonetheless be observed very clearly [Fra03]. Obviously, the usability of component clusters for applications on large data sets that require small clusters is limited.

Furthermore, there are applications like the library data set where disjunctive clusters might not be the optimal choice. In such data sets "bridges" may link document groups belonging to different subject groups. As an example, consider a book B on statistics for psychologists. This is a bridge between the groups "psychology" and "statistics"; these groups shall be disjunctive for the sake of argument. In addition, let us assume that the shares of psychological and statistical contents in the book B are equal. A disjunctive clustering has four possibilities of coping with this situation:

- Treat B as an outlier and keep the clusters for the two subjects separate. This does not optimally reflect the contents of the book and its connections to both subject groups.

Table 3.2: Component clusters for the example

level	clusters
1	$\{A, B, C, D, E, F, G\}$
2	$\{A, B, C, E, F, G\}, \{D\}$
3	$\{A, B, C, G\}, \{D\}, \{E\}, \{F\}$
4	$\{A\}, \{B\}, \{C, G\}, \{D\}, \{E\}, \{F\}$

- Link B to both groups, thus constructing one big cluster. This has the disadvantage that two different matters – psychology and statistics – are joined.
- Assign B to one of the groups. Since B is assumed to be positioned "in the middle" between the two subject clusters, the choice of the cluster is an arbitrary decision that generates a considerable error.

A non-disjunctive clustering, on the other hand, could treat B as follows:

- B's cluster contains some – though not necessarily all – documents from both groups.
- The cluster for a document about "psychology" includes B, but no document from the "statistics" group, and vice versa.

This reflects the actual situation much better which motivated the development of walk context clusters.

Walk Context Clusters

The main reason for the large size of the component clusters are bridge elements like the exemplary book on statistics for psychologists mentioned above. It is sufficient to have one walk cross this bridge to join the two otherwise disjunctive clusters. In other words, one strong, but possibly random edge that is not representative for the cluster is sufficient to link two clusters that should not be linked. Furthermore, the additional question remains how to treat these bridge elements in a semantically sensible way as discussed above.

The idea of walk context clusters is thus to only use the context of the walks in which the objects appear directly. In order to generate the cluster for an object, all walks are considered that contain that object in a step with a level higher than the desired cutoff level. The cluster is then given by all objects also occurring in these walks at a level higher than the cutoff.

The walk context clusters are small and have a high precision, as we will see in the results section (3.3).

This is due to the fact that bridge elements have but a limited influence on the cluster construction: First, the probability of selecting a way across the bridge element is usually less than one, i.e. the bridge is not necessarily included in the cluster. Second, even if the random process decides to include a bridge element in one of the walks, the bridge element does not completely link the two otherwise independent clusters. Only the elements that are reachable from the bridge connection are eligible to be included in the cluster, and even this takes place only with a limited probability.

A second aspect in the construction of an alternative cluster construction method is the fact that, when dealing with graphs with locally varying density, the length of the walks also differs considerably. Remember the basic idea of cluster construction: The later in a walk a pair of nodes is visited, the higher is the tendency of those nodes to belong to the same cluster. In addition, the precision and significance of the clusters depends on the length of the walks contributing to them as we will see below. Component clusters only take into account the absolute position of a pair of nodes in its walk. This assumption is unproblematic as long as the data set under scrutiny is relatively homogeneous in terms of the number of objects that can be reached from each node, the density or number of incident nodes. This is for instance the case for the geometric interpretation of an object set with a pairwise distance function that is defined sensibly (i.e. $0 < d_{ij} < \infty$) for all pairs of nodes.

If, on the other hand, the data set is represented as a graph with similarities as edge weights, and if the density, i.e. the number of incident edges for different nodes, varies, considerably different walk lengths may be the consequence. Thus walks in denser parts tend to be longer than their counterparts in sparser areas of the graph even though the similarity of their last node pairs may be comparably high. When clusters with high precision are sought, this leads to a systematic underrepresentation of the sparsely populated areas, where there is no possibility for the development of long walks. As a result, pairs with high similarity do not show up as clusters if they belong to a region with a low density. This is counterintuitive, since for clustering, the number of neighbors should not play an important role, only their association – in terms of similarity or distance – to each other.

Furthermore RRW clustering has a local view on the data set as will be seen in section 3.2.2, and the applications described in section 3.3 need this local perspective rather than a global one. This problem is not attacked by the component clusters since it did not occur in Schöll and Schöll-Paschinger's setting. But for data sets with varying local densities a relative measure for the position of node pairs may give more intuitive results than an absolute one.

The first and most obvious remedy for this problem is the relative level defined as

$$l = \frac{\text{step number}}{\text{walk length}} \tag{3.6}$$

While this measure levels out the differences in the walk lengths, it does so in too thorough a way. Although the relative position of a node pair is more important than the absolute one, the total walk length conveys information about the quality of the information contained in the whole walk as mentioned above: The longer a walk, the higher is the resolution of the level measure. For example, l will give equal importance to the last – and only – step of a one-step walk and

to the last step of a ten-step walk. While the first relies on a stochastic choice between the neighbors of i_0, the latter quite reliably designates two members of the same cluster. A further advantage of longer walks is their better resolution: The more steps a walk has, the more levels are available in the resulting entries of the dendrogram. This is why two further level measures were devised that represent a compromise between absolute and relative position in the walk:

$$l^+ = \frac{\text{step number}}{\text{walk length+1}} \tag{3.7}$$

$$l^- = \frac{\text{step number} - 1}{\text{walk length}} \tag{3.8}$$

As the walk length approaches infinity, both only strive asymptotically to one for the last, i.e. most meaningful step of a walk. The differences between l^+ and l^- can mainly be seen in the first part of walks and for short walks, where l^+ punishes short walks less severely than l^-. As can be seen from Figs. 3.3 and 3.4, the difference becomes negligible for longer walks:

$$\begin{aligned}
\lim_{\text{length}\to\infty} l^+ - l^- &= \lim_{\text{length}\to\infty} \frac{\text{step}}{\text{length}+1} - \frac{\text{step}-1}{\text{length}} \\
&= \lim_{\text{length}\to\infty} \frac{\text{step}\cdot\text{length} - (\text{length}\cdot\text{step} + \text{step} - \text{length} - 1)}{\text{length}(\text{length}+1)} \\
&= \lim_{\text{length}\to\infty} \frac{\text{length} - \text{step} + 1}{\text{length}(\text{length}+1)} \\
&= \lim_{\text{length}\to\infty} \left(\frac{1}{\text{length+1}} + \frac{1-\text{step}}{\text{length}(\text{length}+1)} \right) \\
&= 0
\end{aligned}$$

For practical purposes, both measures are practically equivalent [FGS05] in terms of the quality of the resulting clusters.

For the example containing the walks $DFBCG$, EGC, BGE, ACG, BCA, $CBACG$, $GBCA$, and $DACG$ and using the l^+ measure, we obtain the clusters shown in Tab. 3.3. Since the clusters are no longer disjunctive, the clusters for all nodes at levels $0.2, 0.5$, and 0.8 are listed.

As can be seen, the high-level clusters only contain the highest-ranked link between nodes C and G. Lowering the level, the central component $\{A, B, C, G\}$ is detected; additionally, due to their strong link, E is added to the cluster for G, but not for A, B, or C since it is not well connected to these latter nodes. D and F remain singletons at this level since their incident edges only bear relatively small weights.

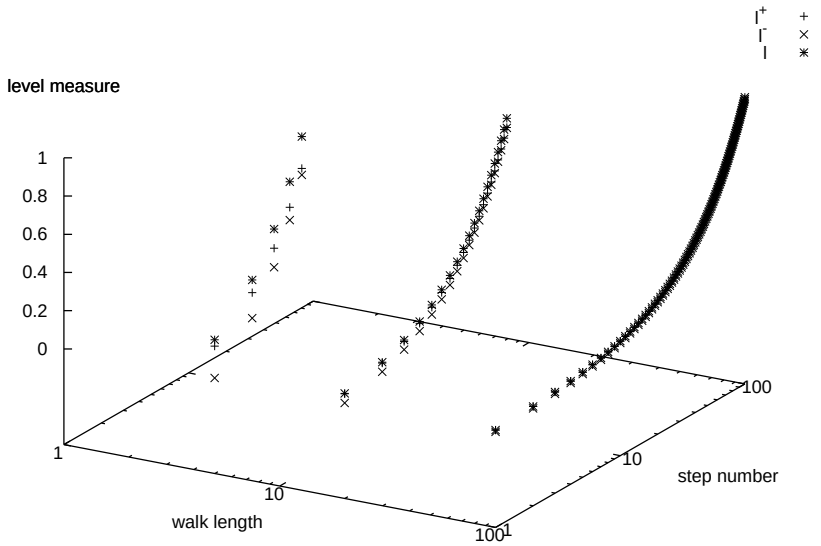


Figure 3.3: Behavior of l, l^+, and l^- for walk lengths 5, 20 and 100 and the respective step number

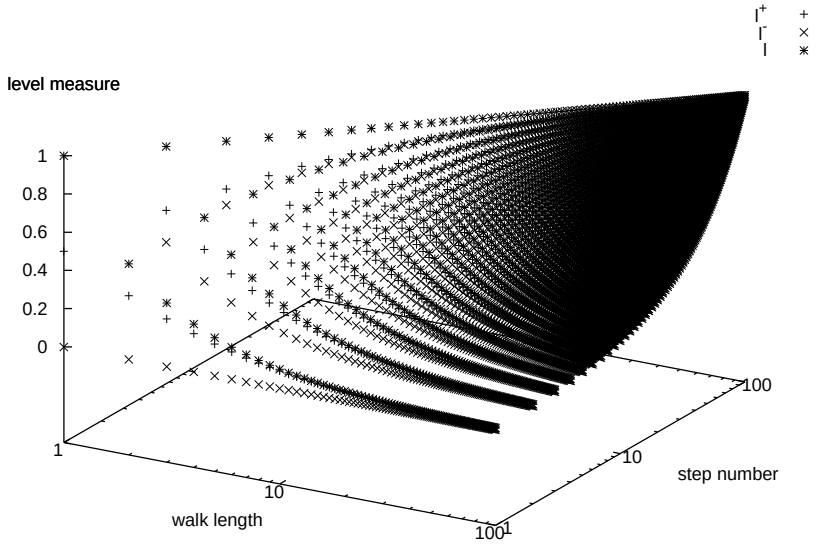


Figure 3.4: Behavior of l, l^+, and l^- for different combinations of walk length and step number

Table 3.3: Walk context clusters for the example (Fig. 3.2)

l^+	A	B	C	D
0.2	$\{B, C, D, G\}$	$\{A, C, D, E, F, G\}$	$\{A, B, D, E, F, G\}$	$\{A, B, C, F, G\}$
0.5	$\{B, C, G\}$	$\{A, C, G\}$	$\{A, B, G\}$	$\{\}$
0.8	$\{\}$	$\{\}$	$\{G\}$	$\{\}$

l^+	E	F	G
0.2	$\{B, C, G\}$	$\{B, C, D, G\}$	$\{A, B, C, D, E, F\}$
0.5	$\{G\}$	$\{\}$	$\{A, B, C, E\}$
0.8	$\{\}$	$\{\}$	$\{C\}$

3.2 Properties of the Method

As we have seen, the original formulation of the method by Schöll and Schöll-Paschinger [SP02] as a stochastic process on the *nodes* of the similarity graph leads to a finite, reducible, second-order Markov chain. With a finite number of nodes and a similarity that increases in each step, the end of a walk is definitely reached in finite time. Furthermore, the chain is reducible since, after starting a walk, it becomes soon impossible to reach all other nodes of the graph – in most cases, this happens with the selection of the first node, but in rare cases also later, but only if the graph is connected and it is possible to reach the start node again via a cycle. Finally, the chain is of second order because the probability distribution over possible successors depends not only on the current node, but also on its predecessor.

In contrast, the alternative *step*-based model presented in the last section leads to a Markov chain with the following properties: It is of first order since the next step only depends on the last step. Due to the concatenation of the walks using a transition state Ω the chain is infinite and, as all states can be reached from Ω per construction, it is also irreducible.

There are however two problems with this approach: First, the state space's size is the square of the original model's state space size, and the process has an even sparser transition matrix than the latter, which enlarges computational requirements for the analysis of the algorithm – but note that the time complexity of the actual algorithm is of course not affected. Second, as shown above, the model converges only asymptotically against the method presented, since the implementation incorporates a deterministic choice of the walk's starting point whereas in the probability model analyzed here it is stochastic. However, given the large number of walks, this is not of real concern here.

3.2.1 Complexity

One aspect that the applications presented in section 3.3 have in common is that they operate on a quite large data set. In this context, the computational complexity of RRW clustering is of considerable interest.

Let us first consider the walks themselves. For an object set of size n, Schöll and Schöll-Paschinger have shown that the expected length of one walk is $O(\log n)$, especially the logarithm with base two ($\log_2$) seems to be a good estimator. This is valid for every object set where the number of neighbors depends on n, like, for example, the original case where every pair of objects has a finite distance associated with it. However, there are cases imaginable where the size of the neighborhood does not depend on n but is rather bounded by a constant c. For an example, think of the network formed by connecting each person of a given group with his or her c best friends.

Theorem 3.2.1 *If the size of the neighborhood of every node can be limited by a constant c, the expected length of all walks is $O(\log c) = O(1)$.*

This will be shown here for the case where the distributions of the neighbor similarities in each node are identical, for instance, are all exponential. The proof shows that the expected size of the successor set is halved in each step. If this is the case, the repeated halving of the set's size leads to a classical divide-and-conquer scenario that has a complexity of $O(\log c)$. We denote the number of nodes incident to the m-th node in the walk as n_m. The n_m are assumed to independent identically distributed random variables. Obviously, $n_m \leq c$, and we define μ_n as the expected value $E(n_m)$.

For the proof, imagine that for the sake of picking a successor, the neighbors of the current node are sorted in ascending order and numbered according to their similarity $1, \ldots, n_m$. Picking one of the n_m neighbors is then identical to choosing its index or position number in the list of neighbors, i.e. drawing a random variable i from the uniform distribution over the set $[1, n_m]$.

We can expect that the higher part of the ordered neighbor set contains an average of $O(\frac{c}{2^{m-1}})$ elements that have a higher similarity than the one used in the current step. Working with this set with size $O(\frac{c}{2^{m-1}})$, we once again obtain the successor by picking from a uniform distribution over $[1, O(\frac{c}{2^m})]$, halving in each step the expected size of the successor set. This is a typical structure found for instance in divide-and-conquer algorithms, leading to a runtime of $O(\log c)$. The argumentation remains valid even if there are ties between neighbors. In this case, the size of the successor set is further reduced which is compatible with the expectation to find at most $O(\frac{c}{2^k})$ admissible successors in the k-th step.

The proof is achieved via induction: The induction starts by establishing the set T_{e_0} from the neighbors of node i_1. Thus we obtain

$$E(E(|T_{e_0}|)) = E\left(\sum_{i=1}^{n_0} \frac{n_1}{n_0}(i-1)p(e_0 = i)\right) \tag{3.9}$$

To obtain the expected value of $|T_{e_0}|$, we sum over all possible choices for the successor of the initial node times the probability that this choice is realized. If n_0 and n_1 were equal, the number of nodes admissible after the choice i would be $i-1$. Since this is not the case, but since the distributions are equal, we calculate the quantile $\frac{i}{n_0}$ in which the choice lies and then transform it to match the number n_1 of neighbors for the next node. We substitute $p(e_0 = i)$ with the concrete probability $\frac{1}{n_0}$ for the event that one of the initial node's neighbors is chosen:

$$= E\left(\sum_{i=1}^{n_0} \frac{n_1}{n_0}(i-1)\frac{1}{n_0}\right)$$

Substituting i to start from 0 gives:

$$= E\left(\sum_{i=0}^{n_0-1} \frac{n_1}{n_0} i \frac{1}{n_0}\right)$$

We can transform the sum $\sum i$ to $\frac{n_0(n_0-1)}{2}$:

$$= E\left(\frac{n_1}{n_0^2}\frac{n_0(n_0-1)}{2}\right)$$

Reducing the term by n_0 leads to

$$= E\left(\frac{n_1(n_0-1)}{2n_0}\right)$$

Simplifying gives:

$$= E\left(\frac{n_0 n_1 - n_1}{2n_0}\right)$$

Using the linearity of the expectation, we get

$$= E\left(\frac{n_1}{2}\right) - E\left(\frac{n_1}{2n_0}\right)$$

Finally, substituting the expected values gives us

$$= \frac{\mu_n - 1}{2}$$

which proves the induction start. Consequently, we can expect that the size of the first step's successor set is on average $O(\frac{\mu_n}{2})$. Thus, given that all nodes have the same distribution over their neighbors – though not necessarily the same number of them – this means that the expected set size is halved as we expect the node chosen in the first step to lie in the middle of this distribution. In the induction step m, of the original n_{m-1} neighbors, we expect $O(\frac{n_{m-1}}{2^{m-1}})$ to be admissible successor candidates for the step. Consequently, the expected size of the successor set after the step m is

$$E(E(|T_m|)) = E\left(\sum_{i=1}^{\frac{n_{m-1}}{2^{m-1}}} \frac{n_m}{n_{m-1}}(i-1)\frac{1}{\frac{n_{m-1}}{2^{m-1}}}\right) \tag{3.10}$$

Once again, we have the (adapted) number of legal successors for every possible i times the probability for this i which is given by a uniform distribution over the $\frac{n_{m-1}}{2^{m-1}}$ legal successors we expect to have in the m-th step. The first term in the sum can be extracted in a fashion analogous to the one used in the induction start, equally we obtain $\frac{n_{m-1}}{2^{m-1}}(\frac{n_{m-1}}{2^{m-1}}-1)$ for the sum $\sum i$:

$$= E\left(\frac{n_m}{n_{m-1}\frac{n_{m-1}}{2^{m-1}}}\frac{\frac{n_{m-1}}{2^{m-1}}(\frac{n_{m-1}}{2^{m-1}}-1)}{2}\right)$$

$\frac{n_{m-1}}{2^{m-1}}$ cancels, leaving

$$= E\left(\frac{n_m(\frac{n_{m-1}}{2^{m-1}}-1)}{2n_{m-1}}\right)$$

expanding the term by 2^{m-1} gives

$$= E\left(\frac{n_m 2^{m-1}(\frac{n_{m-1}}{2^{m-1}}-1)}{2^m n_{m-1}}\right)$$

distributing the contents of the inner parentheses, we obtain

$$= E\left(\frac{n_m n_{m-1} - n_m 2^{m-1}}{2^m n_{m-1}}\right)$$

and finally, splitting and reducing the fraction,

$$= E\left(\frac{n_m}{2^m} - \frac{n_m}{2n_{m-1}}\right)$$

which, substituting the expectations, gives

$$= \frac{\mu_n}{2^m} - \frac{1}{2}$$

In other words, the set size from step $m-1$ to step m has halved, which gives us the desired divide-and-conquer situation from which follows a complexity of $O(\log c)$ for $c \geq n_m$ for all objects m.

If we start, as proposed by Schöll and Schöll-Paschinger, a constant number of walks per node, this leads to a total complexity of $O(n \log n)$ for the case of neighborhoods only limited by the total object set size and $O(n)$ for object sets with constant-bounded neighborhood sizes.

The computational complexity of the different cluster construction methods differs. Component clusters have a complexity of $O(n^3)$ for the naïve approach in the worst case. Using the fact that the cluster construction basically consists of the detection of components in a graph, the complexity can be reduced to $O(n + n^2) = O(n^2)$: As Tarjan [Tar72] has shown, finding connected components has a complexity of $O(n + |E_k|)$ where $|E_k|$ is the number of edges in the graph H_k. Furthermore $|E_k| = O(n^2)$, which results in the complexity given above.

In order to derive the complexity of the walk context clusters, a storage of the walks using an efficient hash table is assumed, such that each element can be retrieved in constant time. Every walk among the nl walks started – where l is the number of walks departing from each of the nodes – has an expected length of order $O(\log n)$ as seen above, leading to a total expectation of $O(n \log n)$ for the number of entries generated by all walks. Furthermore, the average number of visits to each of the n nodes is $O(\log n)$. Analogously, for a bounded neighborhood size, the number of visits is $O(1)$. This implies that the number of walks in which the node in question is included is $O(\log n)$, each of them containing $O(\log n)$ entries on the average. In total, $O(\log^2 n)$ entries must be retrieved for the cluster construction of one node, or $O(n \log^2 n)$ for all clusters. In the case of bounded neighborhood sizes, these numbers reduce to $O(1)$ and $O(n)$, respectively.

3.2.2 Asymptotic Behavior of the Walk Process

In this section, the asymptotic behavior of the random walk process underlying the cluster algorithm will be scrutinized [FGS07a]: What behavior does this process display as the number of walks approaches infinity? In that case, every possible path along the nodes of the similarity graph is taken almost surely, i.e. with probability one. In the following, we assume that every admissible combination of nodes that forms a walk occurs at least once in the infinite process.

The objective of this analysis is to be able to characterize the behavior of the walk process in the asymptotic view, which will finally allow to analyze the properties of the clusters at different "levels" of randomness. On the one side of the scale, where the number of walks approaches zero, the stochastic influence is very strong. The results between consecutive clusterings on the same data set may vary considerably if just one walk is executed. Clearly, this scenario strongly violates

the stability requirement for cluster algorithms stipulated in section 2.3. On the other hand, there are two arguments against too high a number of walks: One of them is operational, since the time needed for the execution of the algorithm increases linearly with the number of walks. The other one is conceptual: It may be desirable to include certain nodes into a given cluster only with a probability that is smaller than one. In order to do so, it is useful to know the characteristics of the process for different numbers of walks including infinity. A sub-aspect in these deliberations is also the knowledge of the point at which behavior changes between "useless random", "useful random" and quasi-deterministic. In the following, the latter case is considered as a starting point, while the other two remain as further research topics.

As a first step the convergence of the occurrence probability to 1 will be shown:

Theorem 3.2.2 *If the underlying restricted random walk process is granted an unlimited number of walks, the probability of a walk occurring at least once is 1.*

For the proof, we need the probability that a specific walk is taken. As a side remark, it can be assumed that the walks take place on a finite object set. For a walk $w = (e_0, \ldots, e_k)$, the probability for its first edge is $P(e_0|\Omega) = \frac{1}{|V|\text{deg}(i_0)} > 0$ as given by Eq. (3.3) on page 68. For every following edge, the probability is $P(e_m|e_{m-1}) = \frac{1}{|T_{e_{m-1}}|} > 0$, where $T_{e_{m-1}}$ is given by Eq. (3.4). In total, the probability for the walk w is

$$P(w) = \frac{1}{|V|\text{deg}(i_0)} \prod_{m=1}^{k} \frac{1}{|T_{e_{m-1}}|} > 0$$

Therefore, the probability that w is not taken when only executing one walk is $0 \leq 1 - P(w) < 1$ and thus the probability that it is not taken during n walks is $(1 - P(w))^n$. As n goes to infinity, we obtain $\lim_{n\to\infty}(1 - P(w))^n = 0$ for the probability of no occurrence and can conclude that the probability of the inverse event, i.e. w occurring at least once is $1 - 0 = 1$ which proves the claim.

To analyze the process – that, as we have seen in section 3.1.1, is a Markov chain – for an infinite number of walks, consider a directed, unweighted graph G' constructed in the following way: All states of the Markov Chain (i.e. all pairs of nodes connected by an edge in the original similarity graph $G = (V, E, \omega)$) constitute the set of nodes V'. There is an arrow from node B to node C in G' iff C is an admissible successor state of B in the original process. For the purpose of this analysis, the state Ω is split into two nodes of the graph, A (capital alpha) and Ω. All nodes of G' can be reached from the starting node A and all nodes without

successor have an arrow pointing to the node Ω. Formally, this graph G' is defined as

$$G' = (V', E') \tag{3.11}$$

with

$$V' = E \cup \{A, \Omega\} \tag{3.12}$$

and

$$\begin{aligned} E' &= \{((i,j),(j,k)) \in E \times E | \omega_{jk} > \omega_{ij}\} \\ &\quad \cup \{A\} \times E \\ &\quad \cup \{(i,j) \in E |\ \nexists (j,k) \in E : \omega_{jk} > \omega_{ij}\} \times \{\Omega\} \end{aligned} \tag{3.13}$$

It is on this graph that the asymptotic behavior of the restricted random walk process is studied using a combination of shortest and longest path algorithms. For the latter it is important to note that G' is cycle-free, such that a finite longest path can always be found. A path is defined in this context as follows:

Definition 3.2.1 (path) *A path on G' is the succession of states (that may be edges of another graph) of a Markov chain or a part thereof.*

For the basic idea, consider Fig. 3.5 and remember the level definitions from 3.1.2. The following considerations hold for all of the level measures introduced in section 3.1.2. The characteristics of the clustering depend on the position of the node pairs, and, in the case of walk context clusters, on their combination inside the walks. For any node B, the walks visiting it can be split in two parts, the one before crossing B, and the one afterwards. The position of a node is computed using a separate analysis for the first part of the walk and the second one. Let a be

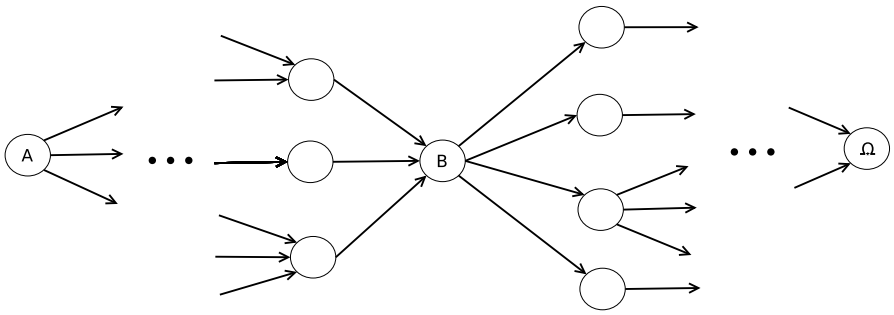


Figure 3.5: Different paths meeting and separating at node $B \in V'$

the length of the longest path leading from A to B and o the length of the shortest walk from B to Ω. The resulting level in the case of the l^+measure is

$$l^+ = \frac{a}{a+o} = \frac{1}{1+\frac{o}{a}}$$

It is clear that for any of the measures l, l^+ and l^-, the level is maximal when o is minimal and a is maximal for the given node of G'.

For the asymptotic case, the following algorithm is used for determining the cluster level of every node in G', i.e. the maximum level at which the objects contained in the node are still assigned to the same cluster. The graph is cycle-free, since per construction, due to the strictly increasing similarities on the edges, a node B in G' cannot be visited more than once per walk. As a consequence, Dijkstra's algorithm [Dij59] can be used both for calculating the longest and the shortest paths for any node in G'.

For the computation of the a, the following algorithm is used, with $\mathcal{P}_\mathcal{B}$ the set of predecessors of B:

$\mathcal{F} = \{A\}$
$a_A = 0$
While ($\mathcal{F} \neq E \cup \{A, \Omega\}$):
  Select state B with $B \notin \mathcal{F}$ and $\mathcal{P}_B \subseteq \mathcal{F}$
  $a_B = \max_{C \in \mathcal{P}_B}\{a_C\} + 1$
  $\mathcal{F} = \mathcal{F} \cup \{B\}$
End

o is computed in an analogous fashion, with $\mathcal{S}_B$ the set of successors of B:

$\mathcal{F} = \{\Omega\}$
$o_\Omega = 0$
While ($\mathcal{F} \neq E \cup \{A, \Omega\}$):
  Select state B with $B \notin \mathcal{F}$ and $\mathcal{S}_B \subseteq \mathcal{F}$
  $o_B = \min_{C \in \mathcal{S}_B}\{o_C\} + 1$
  $\mathcal{F} = \mathcal{F} \cup \{B\}$
End

With these maximum levels computed, it is possible to characterize the behavior of the walk process. A first example is shown here; for the future, a theoretical analysis of the cluster construction process is planned. The result resembles that of a single linkage clustering, however, the clusters are not identical on all levels of the cluster hierarchy.

For this example, the Deep South data set from [DGG48] is used. The data

Table 3.4: The similarity matrix derived from the Deep South data set by Davis et al. [DGG48]

IDs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	-	6	7	6	3	4	3	3	3	2	2	2	2	2	1	2	1	1
2	6	-	6	6	3	4	4	2	3	2	1	2	2	2	2	1	0	0
3	7	6	-	6	4	4	4	3	4	3	2	2	3	3	2	2	1	1
4	6	6	6	-	4	4	4	2	3	2	1	1	2	2	2	1	0	0
5	3	3	4	4	-	2	2	0	2	1	0	0	1	1	1	0	0	0
6	4	4	4	4	2	-	3	2	2	1	1	1	1	1	1	1	0	0
7	3	4	4	4	2	3	-	2	3	2	1	1	2	2	2	1	0	0
8	3	2	3	2	0	2	2	-	2	2	2	2	2	2	1	2	1	1
9	3	3	4	3	2	2	3	2	-	3	2	2	3	2	2	2	2	1
10	2	2	3	2	1	1	2	2	3	-	3	3	4	3	3	2	3	2
11	2	1	2	1	0	1	1	2	2	3	-	4	4	3	3	2	3	2
12	2	1	2	1	0	1	1	2	2	3	4	-	6	5	3	2	3	2
13	2	2	3	2	1	1	2	2	3	4	4	6	-	6	4	2	3	2
14	2	2	3	2	1	1	2	2	2	3	3	5	6	-	4	1	2	2
15	1	2	2	2	1	1	2	1	2	3	3	3	4	4	-	1	1	1
16	2	1	2	1	0	1	1	2	2	2	1	2	2	1	1	-	1	1
17	1	0	1	0	0	0	0	1	1	1	1	1	1	2	1	1	-	2
18	1	0	1	0	0	0	0	1	1	1	1	1	1	2	1	1	2	-

describe the attendance of 18 women to 14 social events and is contained in appendix B. It is often considered in sociology since there is no unique, indisputable assignment of women to groups as for instance Freeman's meta-study has shown [Fre03].

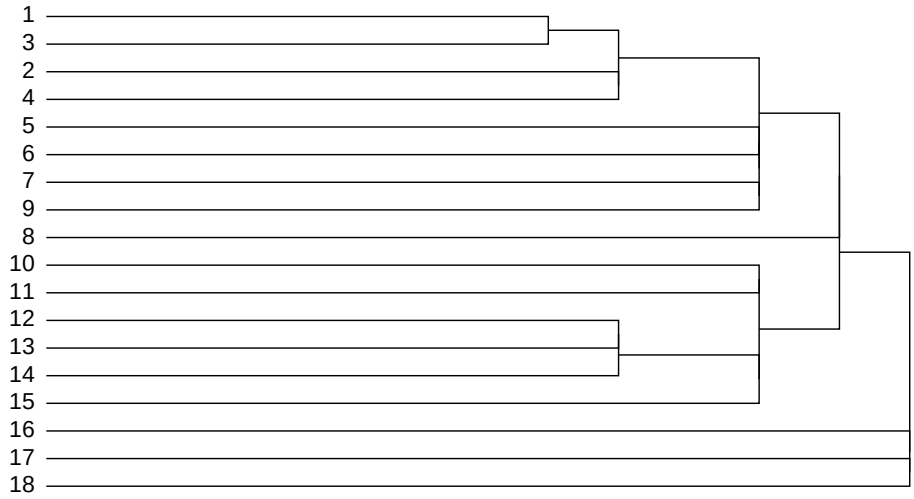


Figure 3.6: The dendrogram for the Deep South data set with single linkage clustering

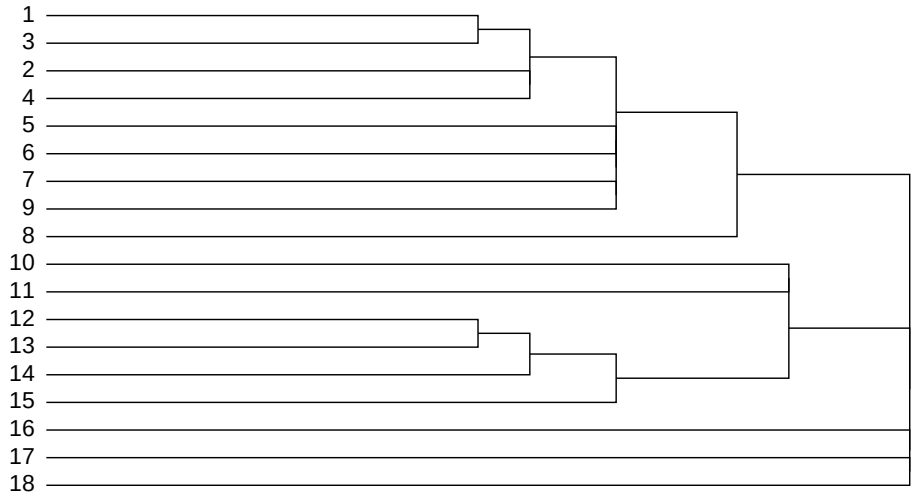


Figure 3.7: The dendrogram for the Deep South data set using RRW clustering and the l^+ measure

It is used here to give a first impression of the typical differences between single linkage clusters and the asymptotic behavior of RRW clusters. The similarity

between two women as given by Tab. 3.4 is coded as the number of events they have taken part in together. The data set was clustered using both single linkage clustering and restricted random walk clustering with the component cluster method and the l^+ level measure; the resulting dendrograms are given in Figs. 3.6 and 3.7. The first impression is that the clusterings have a high similarity, however, there also exist interesting differences.

The first of these effects is due to the limited horizon of the RRW method. Consider the pairs 2/4 and 12/13 in both dendrograms. As can be seen, the pairs are merged on the same level by the SL method: Since they have the same similarity, the global comparison operator of the SL method schedules both pairs for a merge on the same level. RRW, on the other hand, lacks this global view – which incidentally contributes to the smaller run time – and thus decides on the basis of local similarity densities. In our example, the different densities result in a maximal walk length of five for the pair 2/4 and six for 12/13. In a sense, the RRW algorithm offers an additional level of differentiation. As shall be seen in section 3.3.3, this is an important feature if the data set is sparse.

On the other hand, there are pairs with different similarities in the data set that are merged at the same level by the RRW method, for instance the pairs 1/3 and 12/13. This effect is due to the fact that the similarity structure around those pairs is similar, though on different levels. This property of the RRW method is useful in applications where it is more important to detect local maxima than global ones.

3.3 Excursus: Applications

The clustering method presented in this chapter is applicable to every object set with members that have a similarity or distance relation. However, due to its relatively low computational complexity (cf. section 3.2.1) it is especially with large data sets that the algorithm shows its strengths. This is why the algorithm was originally chosen for the analysis of library usage histories as presented in the following section where it was used for the purpose of indexing library corpora and the generation of recommendations.

3.3.1 Library Usage histories

When the following studies were conducted in spring 2005, the online public access catalog (OPAC) at the university library in Karlsruhe, Germany, had usage histories on 1.8 million documents out of which about 800,000 have been visited often enough to include them in the clustering. The average degree of a node in this data set is about 39; this implies that the data set comprises nearly 36 million weighted edges. Let us start with the description of the data set before detailing

the two applications, (semi-) automated indexing and recommendation generation, for which RRW clustering was used in this domain.

The usage histories are generated by users of the library OPAC browsing detail web pages of documents. Each visit is counted as an occurrence of that document. All documents viewed by the same user in one session are considered as cross-occurrences. These cross-occurrences between documents are summed up and stored in raw baskets like the one shown in appendix A. It contains the identifier for a document (the last line) plus all other documents, along with their cross-occurrence frequencies. The preprocessing that is necessary to extract these sessions from the web server logs is described e.g. in [GSHNT03] and will not be detailed here.

Since raw baskets stored in files represent a performance bottleneck when working with an object set of this size, the data is transferred to a data base, preserving all information from the baskets.

In order to obtain a transition matrix, the similarity between two objects is defined as their number of cross-occurrences. This measure has all characteristics of a similarity measure as given by definition 1.1.10 except the limitation to the interval $[0, 1]$.

It is important to note that this similarity measure is not based on any kind of content analysis of the full texts or even the titles. This differentiates the approach from many others used in information retrieval [CPS98, LH03, Seb02, SFFE01, Yan99]. A full text analysis is simply not possible in a conventional or hybrid library, since some or all of the texts are not available in a digital format that would allow their fast analysis. Instead, the method presented here relies completely on the behavior of users following their own interests. This has several advantages. First, the data can be gathered automatically, without intervention of the user. This means that there is no need to set incentives for users to cooperate with the system. Second, systems based on implicit user "cooperation" are less prone to problems concerning incentives like manipulations or free riding as discussed by Geyer-Schulz et al. [GSHJ01] or Nichols [Nic97].

Finally, the analysis of transaction data has a tradition in marketing and is used to predict repeat buying behavior [Ehr88] or cross-selling potentials. In the latter case, results show that there is a high correlation between cross-occurrences and complementarity of products. As Samuelson [Sam38, Sam48] states, choice behavior reveals the users' preferences.

3.3.2 Indexing Library Corpora

This application was motivated by the situation of the catalogues of the university's library at Karlsruhe as well as of many other libraries in Germany. As a random sample drawn from the catalogue showed [FGS04], about 30% of the

documents were not indexed with any keywords. Since manual indexing is a time-consuming and therefore expensive activity, there is little chance that these parts of the corpus will ever be indexed by hand. The problem is thus that parts of the collection cannot be found when using an index search. This is a problem for many libraries today, independent of the nature of their content.

While solutions have been proposed for digital libraries based on an analysis of the full text or an abstract [Yan99, Seb02, SFFE01], this is not a viable solution for libraries like the one at Karlsruhe that for the most part contains conventional documents like books and journals in paper form. The solution should be usable for libraries independent of the representation of their content.

As a possible solution, the use of restricted random walk clustering for automated or semi-automated indexing on the basis of usage histories was investigated. The idea is that if the clusters obtained in this way contain similar documents, then there should be a high similarity in the keywords of the cluster members, too. Consequently, for a document without manually assigned keywords it is reasonable to consult the distribution of the keywords assigned to the other members of its cluster.

As a first step, restricted random walks are executed on the document set based on the raw baskets containing the usage histories. After performing all walks, the cluster for each document without keywords is constructed, and the occurrence frequencies of the keywords assigned to documents in the respective cluster are counted. Obviously, the higher the fraction or number of documents sharing a keyword in a cluster, the higher is the probability that this keyword also fits for the document without index terms that is the center of the cluster.

Formally, the relevance of a keyword k for a document i can be defined in several ways. If $f_l(k,i)$ is the number of times k occurs in i's cluster at level l, and $t_l(i)$ is the size of i's cluster at level l, the following measures for significance are conceivable: First, the absolute frequency of term occurrence

$$\text{sig}_l^{\text{abs}}(k,i) = f_l(k,i) \tag{3.14}$$

second, its relative frequency

$$\text{sig}_l^{\text{rel}}(k,i) = \frac{f_l(k,i)}{t_l(i)} \tag{3.15}$$

or an adjusted measure,

$$\text{sig}_l^{\text{adj}}(k,i) = \frac{f_l(k,i)-1}{t_l(i)} \tag{3.16}$$

The theoretic drawbacks of the first two measures are obvious. The absolute frequency does not take into account the total number of documents in the cluster.

Consequently, the event "10 documents out of 100 have keyword k" will be rated higher than "9 out of 9 documents have keyword k" while intuitively, the second is more meaningful. The relative measure is blind in another way in that it does not consider the absolute number of documents supporting a keyword. So, no matter if the cluster consists of one or 10 documents the measure will yield the same result although the first may solely be based on a single document that has wrongly been assigned to the cluster.

Thus, in analogy to the l^- measure from section 3.1.2, the adjusted measure was introduced that accounts for both important factors, cluster size and relative support for a keyword inside the cluster. For large cluster sizes, it converges to one.

The indexing was tested on two different classification schemes both in use at the library. The first one is based on the SWD Sachgruppen indexing scheme published by the Deutsche Bibliothek [Kun03]. It is a numerical, hierarchical classification with four levels. Due to the already mentioned sparse manual classification, only the two highest levels were used.

For the SWD Sachgruppen, in addition to a manual evaluation [FGS04], we conducted an automated evaluation for 15,000 documents that already had manually assigned keywords – of course without making use of the keywords in the recommendation generation process. The cutoff level of the clusters was set to 0.5; sig_l^{adj} was used as significance measure with a significance threshold of 0.27. The evaluation measure was the precision of the keywords

$$\text{prec} = \frac{\text{number of correctly assigned keywords}}{\text{total number of assigned keywords}}$$

where the correctness of a keyword was judged according to whether the keyword had also been manually assigned to the document in question. With the parameters given, it was possible to obtain a precision of 77% while generating keywords for 66.8% of the documents. This seems to be an acceptable compromise for the ever-recurring problem of information retrieval: Precision versus recall.

A further evaluation was undertaken on the keywords from the second classification scheme in use. In this case, real keywords like "differential equations" are assigned to the documents. Reference librarians were asked to judge the quality of the keywords that the algorithm generated for 212 documents at $l^+ = 0$. At that level, nearly 4,000 keywords were generated – keep in mind that a level of zero is the most imprecise level, but also the one with the highest recall. Of these, 365 were classified as correct for their respective document. This data base of evaluation results allowed a more thorough evaluation of the necessary parameters of level, significance measure and significance cutoff. In addition to the three significance measures in [FGS04], more were developed that use for their definition the terms $j \in C_0'(i,k)$ for the documents in i's cluster at level 0 that also have the

keyword k and $l^+_{max}(i,j)$ for the maximum level at which j is still in i's cluster. The measures tested were:

$$\text{sig}^A(k,i) = \frac{1}{t_0(i)} \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.17}$$

$$\text{sig}^{A+}(k,i) = \frac{1}{t_0(i)+1} \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.18}$$

$$\text{sig}^B(k,i) = \frac{1}{f_0(k,i)} \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.19}$$

$$\text{sig}^{B+}(k,i) = \frac{1}{f_0(k,i)+1} \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.20}$$

$$\text{sig}^C(k,i) = \frac{f_0(k,i)}{t_0(i)} \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.21}$$

$$\text{sig}^{C+}(k,i) = \frac{f_0(k,i)}{t_0(i)+1} \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.22}$$

$$\text{sig}^D(k,i) = \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.23}$$

$$\text{sig}^E(k,i) = f_0(k,i) \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.24}$$

$$\text{sig}^{E-}(k,i) = (f_0(k,i)-1) \sum_{j \in C'_0(i,k)} l^+_{\text{max}}(i,j) \tag{3.25}$$

sig^{C+} proved to generate the best results; they are plotted in Fig. 3.8. As can be seen, a precision of 100% is feasible, but only for a small share, about one per cent of the documents. On the other hand, when all keywords are admitted at the lowest level, precision drops to 10%.

3.3.3 Giving Recommendations

Recommender systems – like the well-known example at amazon.com or the recommender in operation at the university's library at Karlsruhe – offer added value for both sides in a (possible) transaction. The customer is offered a possibility to further explore complementary or substitutive products while the vendor can

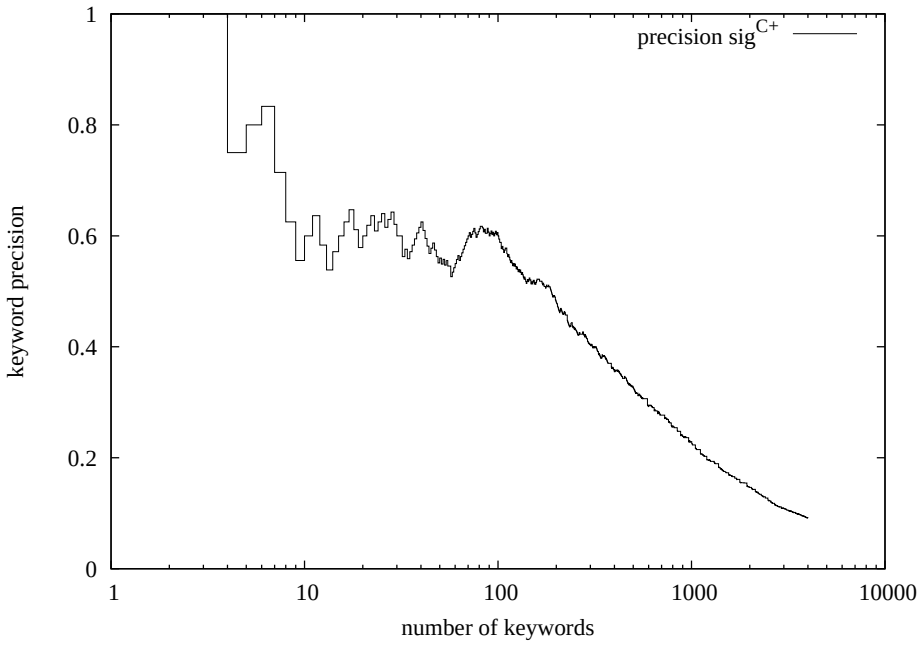


Figure 3.8: Precision versus number of generated keywords (data from [FGS07b])

expect an increase in sales by directing the customer's attention towards complementary products. In a non-profit setting like at the university's library, quality of service and thus user satisfaction can be enhanced. Besides the obvious recommendation for complementary books, it is also conceivable to recommend available substitutes for books that are currently not available.

A general review of recommender systems can be found in the works of Adomavicius and Tuzhilin [AT05], Gaul et al. [GGSHST02], Resnick and Varian [RV97] or Schafer et al. [SKR01]. All implicit recommender systems rely on Samuelson's thesis that observed behavior reveals user preferences [Sam38, Sam48]. Besides the relatively simple system used at amazon.com that consists of recommending all items with a (relative or absolute) cross-occurrence frequency above a given threshold, there are more sophisticated methods like the one used at the university's library at Karlsruhe [GSNT03] that profit from the evaluation of the underlying distribution of the cross-occurrences [Ehr88] to distinguish between random and meaningful cross-occurrences in a more robust way. The difference between the two is that random cross-occurrences are created by independent purchase processes, whereas dependent processes generate meaningful cross-occurrences that can be identified by the framework by testing for outliers.

The use of clustering in order to improve the quality of a recommender system has been proposed in the past. A general review of alternative techniques

for collaborative filtering applications has been given by Griffith and O'Riordan [GO02]. One of the first publications to apply clustering for recommendation generation was written by Borchers et al. [BLKR98] stating that a clustering of the users leads to both increased quality and increased scalability by reducing the user data set's size. Sarwar et al. [SKKR02] analyzed the scalability of recommender systems and concluded that clustering users leads to more scalable recommender systems for large user groups. Finally, most recommender systems' matrices are sparse, especially if the system is new or if a new user is to be introduced. Kohrs and Merialdo [KM99] have especially focused on ways to efficiently deal with such matrices.

Getting back to our input data, the observable behavior is driven by the two relations we are searching for: On the one hand, objects in the baskets may be complementary as users search for literature complementing their current selection. On the other hand, substitutes are contained in the data set, because users have browsed all possible literature for a subject before settling on the document best suited for the purpose – or on one of the available ones, if the best book is currently on loan. Thus, by exploiting the usage histories, the recommender is able to offer documents that are related in one of these two ways.

Thus, given the quality of the clusters exhibited at earlier applications, it is possible to use RRW clustering for the purpose of recommendation generation [FGS05, FGSN06, FGS07b]. As argued above, once the clusters are generated, there is not much left to do in order to obtain recommendations, since the cluster members for a given cluster center are already either complements or substitutes. The fact that RRW clustering with the walk context method returns nondisjunctive clusters is an added bonus in this case since as discussed in section 3.1.2, it is e.g. favorable to include psychology and statistics books in the recommendation list of an introductory course in statistics for psychologists, but it is not desirable to include the same psychology books in the cluster for one of the statistics books.

Furthermore, the question of setting the "correct" cutoff level l is solved very elegantly in this context by setting the initial level to an intermediate value and leaving the decision about adjustments to the user since he or she is the only one to know the requirements of the current session. If a broader overview is sought, a low level l can be requested, returning large, but less precise clusters. If, on the other hand, only documents narrowly related to the current one are of interest, this can be achieved with a high value of l.

A prototypical interface exists for displaying the recommendations. Screen shots in analogy to [FGS07b] are given in Figs. 3.9 to 3.11 for the books by Kaufman and Rousseeuw [KR90] as well as by Bock [Boc74]. The top of the web page contains the slider that allows the user to adjust the level to the needs of the current search.

Figure 3.9: Recommendations for Bock [Boc74], high precision

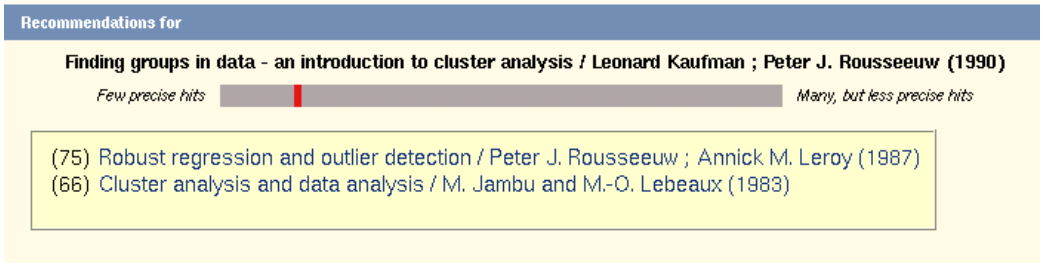


Figure 3.10: Recommendations for Kaufman and Rousseeuw [KR90], high precision

In addition to the above evaluations, two more were conducted for the recommendations. The first one relies on the SWD Sachgruppen classification scheme as benchmark; documents are judged to be correctly in the same cluster if they share at least one keyword. This is a rough criterion for the fitness of the clusters for recommendation purposes; the resulting precision of the clusters for a data set of 40,000 documents is given in Fig. 3.12. Here, it becomes clear that the simple level measure is not suited for the generation of recommendations. Even the most restrictive level only provides a precision of little more than 50% while already including 39,045 documents. l^+ and l^-, on the other hand, give quite similar results, with maybe a small advantage for l^+. The highest precision reachable on this data set is 95.5% using l^+. An additional manual test with 30 documents was undertaken; the authors evaluated the top five recommendations for each docu-

Figure 3.11: Recommendations for Kaufman and Rousseeuw, low precision

ment. The precision on this data set is 78,7%.

In the context of [FGS07b], a comparison with other cluster methods was attempted. There were, however, some difficulties: First, the library data set does not offer vector data, but only similarities. As a consequence, centroid- or medoid-based algorithms like k-means are not applicable to the data. Furthermore, as mentioned in the introduction to the data set, it is large with 0.8 million active objects and 36 million edges which requires efficient algorithms with a time complexity of no more than $O(n \log n)$. But even then, the most simple cluster algorithm – single linkage (SL) – would have taken too long for a significant evaluation. Fortunately, as Gower and Ross [GR69] have shown, all information necessary for SL is contained in the minimum spanning tree of the graph. Thus, in order to find the cluster for a given document at a given level it is sufficient to find the component containing that document in the threshold graph for the level which considerably speeds up the cluster construction. Equally, for the complete linkage cluster algorithm, only documents incident to the cluster seed document must be taken into account since only these have a positive similarity with the seed.

When constructing the clusters for the two books by Bock and Kaufman and Rousseeuw it became already clear that both SL and CL display some serious problems in this setting that RRW does not have. First, the fact that the book by

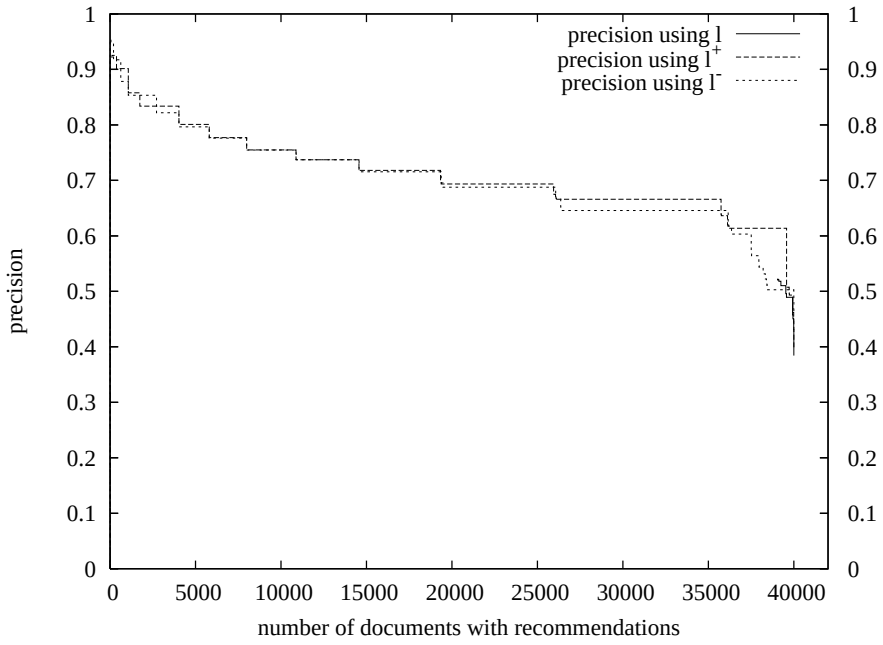


Figure 3.12: Trade-off between recall and precision (data from [FGS07b])

Kaufman and Rousseeuw [KR90] has only four documents out of 50 with cross-occurrence frequencies higher than one, the highest frequency being four, poses a problem for the SL algorithm. Even if the cluster construction is restricted to this highest level, thus only including one of the original neighbors of the book, SL produces a cluster containing more than 250,000 documents. Apart from a cluster of this size being unsuitable for recommendation purposes, the connections of the cluster members are at least questionable given the scant connectivity between seed document and cluster.

CL, on the other hand, does not offer as many levels of differentiation as does RRW. Instead of seven levels, the CL clusters are only differentiated into four groups, which directly follows from the similarity distribution of the neighbors. Additionally, the first three levels only comprised four documents, the rest entered the cluster structure at level one.

For Bock's book [Boc74], the results are less grim. Here, the highest similarity with a neighbor is 19. As a consequence, there is less chaining at the highest level, the SL cluster contains the books by Hartigan, Steinhausen, Mirkin, Bacher, and Gaul also included in the RRW recommendations (cf. Fig. 3.9). The differentiation problem for CL remains, however: There are nine CL levels, and 94 out of 125 documents are only contained in the clusters for the lowest level, while the RRW clusters have 25 different levels.

As a result, it must be stated that both SL and CL are less suited for the generation of recommendations than the RRW method, especially if the similarity distribution is narrow. On the other hand, it must be admitted that no attempts were undertaken to boost the SL/CL performance by e.g. normalizing the similarities, so there might still exist potential for improvements.

3.4 Summary

This chapter has introduced the RRW clustering method along with a short discussion of some properties like complexity or asymptotic behavior. The applications to which the algorithm has been put have been presented, and the evaluations were given that were performed in the context of these applications.

It has been shown that the RRW method is well able to cope with large data sets like the purchase history data set at the university's library in Karlsruhe. The evaluations of the cluster quality are promising as they show a good quality of the clusters detected by the algorithm. This allows us to consider the RRW algorithm as basis for a dynamic cluster algorithm that will be presented in the next chapter.

Chapter 4

Updating Restricted Random Walk Clusters

As the literature review in chapter 2 has shown, dynamic clustering – in the widest sense of the term – has received quite some attention during the last decade. This is due to the fact that more and more large data sets are maintained that are not static but grow and change over time. A good example for this phenomenon is the OPAC usage data set discussed in section 3.3: A living OPAC implies changes in the similarity matrix. New documents are added, and old ones may be removed due to the growth and management of the corpus. As the overview in chapter 2 has shown, the case of growing, and, within limits, shrinking data sets is often considered in the literature. On the other hand there seem to be no algorithms actually capable of dealing in an acceptable way with "mobile" objects, i.e. objects that change their respective distance or similarity dynamically. But this is the even more important case in our scenario when clustering library usage data: Users browse the collection, creating new market baskets that in turn increase the similarity between the documents visited. Furthermore user behavior changes over time.

It follows that our task is to efficiently keep the clusters up to date in the face of insertions and deletions of objects as well as of changing similarities. Clearly, re-executing all walks in order to reflect these changes is not a feasible solution given the running time of the algorithm and the size of the data set. Likewise, it is not a sensible solution given the characteristics of the RRW cluster algorithm as discussed in section 3.2.2. The local perspective of the algorithm implies that changes in the similarity matrix have only local effects on the clusters. In addition, the number of changes per update is relatively small compared to the overall size of the document set, which would even allow the update process to have a slightly higher complexity in terms of the number of objects involved and to still perform better than a complete reclustering. In turn, the quality of the updated clusters

should not deteriorate compared to the original clustering so that a reclustering never becomes necessary.

A further point in favor of an update is the fact that not every change in the matrix has repercussions on the clusters: The changes can be basically reduced to two cases that will be elaborated in the next section: Walks present in the data base may become invalid and have to be removed or new possibilities for walks may be discovered and have to be taken into account. From these considerations, methods will be derived to deal with the basic cases

- insertion of an object,
- removal of an object, and
- update of the similarity between two objects.

4.1 The Basic Update Cases

Before detailing the update algorithm, let us discuss the relevant changes in the similarity matrix and their consequences. For the sake of clarity of the proofs, let us assume for now that the changes are processed separately: As soon as a change occurs, the respective actions are carried out before the next update is integrated into the similarity matrix. Section 4.2 contains a short discussion of the extension for several concurrent updates.

In addition to the set T_{ij}, the set of admissible successors of edge (i, j) from the static scenario in section 3.1.1, we will need the set T'_{ij} that represents the set of admissible successor for the edge (i, j) just as given above, but *after* the update of the similarity matrix, whereas T_{ij} in this context describes the set used in the original setting *before* the update. The term *path* is used in this chapter according to definition 3.2.1, especially in the sense of walk fragments.

For an illustration of the basic cases, consider the graph in Fig. 4.1 where $\omega_{BF} = \omega_{FB} = k$ is a variable weight – or more specifically, similarity – that is subsequently changed by updates of the similarity matrix in order to demonstrate the possible changes to the similarity graph.

Let us say that, in the original data, the value k between nodes B and F equals two. In this case, walks can cross the link FB and continue to either C or A, but a traversal from B to F is only possible if B is the start node of a walk. Now consider the following updates:

1. If, in a first update, the weight increases to $k = 3$, paths that contain the sequence GBF and thus were not feasible before the update become viable. In other words, for walks coming from node G to node B, there is another

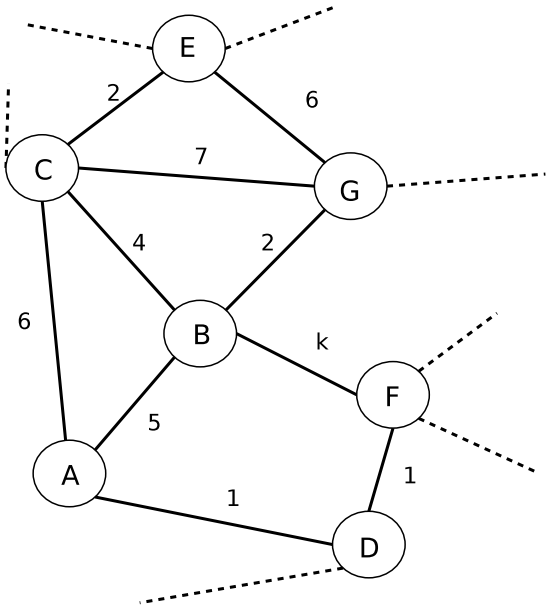


Figure 4.1: Section of a similarity graph

possible edge to choose in addition to BC and BA. The handling of events of this type will be discussed in section 4.1.1.

2. A subsequent update sets $k = 4$. Consequently, all walks containing the sequence FBC become invalid since $\omega_{BC} > \omega_{FB}$ is no longer true. The procedure for handling this class of cases is developed in section 4.1.2.

As mentioned in the introduction to this chapter, not every change in the similarity matrix automatically implies a change in the walks. If, for example, the similarity between C and E changed from 2 to 3, the set of possible walks would remain unaffected by the change, since we only use an ordinal scale for the similarities in order to calculate the transition probabilities.

The insertion of new nodes as well as the removal of existing nodes can be reduced to the cases given above as will be detailed in sections 4.1.4 and 4.1.5.

4.1.1 New Paths

Let us now consider the necessary actions for reacting to case (1) given above. If new possibilities for paths emerge, the algorithm must find all walks that, given the updated similarity matrix, might have included the edge with the increased weight because they include one of its admissible predecessors. In our example, these are all walks containing the edge GB. Before the update, a possible successor

for GB was chosen with equal probability from $T_{GB} = \{BC, BA\}$, i.e. each of theses edges was chosen with a probability of $\frac{1}{|T_{GB}|} = \frac{1}{2}$. If the initial weight on BF had been greater two, T_{GB} would have included the three edges BA, BC, and BF. Consequently, BF would have had a probability of $\frac{1}{3}$ of being chosen, just as BC and BA would have had in that case. It follows that, in order to mimic the output of the original algorithm on the new data set, probability mass must be redistributed from BA and BC to BF such that each of them is expected to occur in $\frac{1}{3}$ of the walks coming from GB. To effectuate this redistribution, the update algorithm considers all walks that cross the edge GB. If BF is an admissible successor, it means that some walks formerly proceeding from GB to either BA or BC should be truncated at this point and continued as normal restricted random walks, but using BF as successor for GB. For each of the walks under consideration, the decision whether to rerun it starting from BF is made based on the probability of

$$P(\text{no change}) = \frac{|T_{GB}|}{|T'_{GB}|} \tag{4.1}$$

that this walk is preserved. If it is not preserved, the entries for this walk in the walk data base are truncated after GB, and the new edge BF is inserted. Afterwards, a restricted random walk is executed as if it had just visited BF, i.e. it continues with the correct step number derived from the walk's first part and chooses a successor from the set T'_{BF}.

Taken altogether, the algorithm assigns a total probability of

$$\frac{1}{|T_{GB}|}\frac{|T_{GB}|}{|T'_{GB}|} = \frac{1}{|T'_{GB}|} \tag{4.2}$$

of being included in a walk after GB to each successor that was already admissible before the update. This is just the probability that this successor would have been assigned had the situation during the initial execution of the walks corresponded to T'_{GB} instead of T_{GB}.

With a probability of

$$P(\text{change}) = 1 - \frac{|T_{GB}|}{|T'_{GB}|} = \frac{|T'_{GB}| - |T_{GB}|}{|T'_{GB}|} \tag{4.3}$$

we discard the rest of this walk and restart it from the node in question, choosing the successor among the $|T'_{GB}| - |T_{GB}|$ new possible nodes with equal probability. Thus, each of these nodes has a total probability of

$$\underbrace{|T_{GB}|}_{\text{number of old successors}} * \underbrace{\frac{1}{|T_{GB}|}}_{\text{P of being chosen from } T_{GB}} * \underbrace{\frac{|T'_{GB}| - |T_{GB}|}{|T'_{GB}|}}_{\text{P of change}} * \underbrace{\frac{1}{|T'_{GB}| - |T_{GB}|}}_{\text{P of being chosen from } T'_{GB}} = \frac{1}{|T'_{GB}|} \tag{4.4}$$

of being picked, which, once again, corresponds to the correct probability when executing the original algorithm on the updated data set.

Formally, the algorithm for new paths over an edge e is as follows:

For all $e' \in E$ with $e \in T'_{e'}$ and $e \notin T_{e'}$:
 For all e'' with $e'' \in T_{e'}$:
  For all walks $(\ldots, e', e'', \ldots)$:
   With probability $1 - \frac{|T_{e'}|}{|T'_{e'}|}$:
    truncate the walk, starting from e''
    insert e at the end of the walk
    continue the walk by picking a successor from T'_e
    follow the original algorithm for the rest of the walk
   End
  End
 End
End

Returning to the example on page 70 with the walks $DFBCG$, EGC, BGE, ACG, BCA, $CBACG$, $GBCA$, and $DACG$, we state that the walk $GBCA$ contains the edge GB for which a new successor is available. As argued above, the set $T'_{GB} = \{BA, BC, BF\}$ now contains the edge BF that was not part of T_{GB}. According to the algorithm, there is a random choice between maintaining the walk in its present form with probability $\frac{2}{3} = \frac{|T_{GB}|}{|T'_{GB}|}$ and truncating it in favor of the new path with probability $\frac{1}{3}$. Let us assume that the walk is truncated: The remainder of the walk, GB, is continued using BF. Since $T'_{BF} = \emptyset$, the walk ends here. This does not change the clustering for the component cluster method since both pairs, BC and BA, from the truncated walk are contained in other walks in equivalent or higher positions and the new pair BF was already present at level two before the update. In contrast, the objects A and B no longer belong to each others' clusters at level 0.5 when using the walk context method (cf. page 77). In return, due to the modified walk, B and F occur now in each others' cluster at level 0.5.

The important advantage of the update procedure given here is that it produces the same results, speaking in terms of probability distributions over the states or edges, as the renewed execution of all restricted random walks on the updated similarity matrix.

Theorem 4.1.1 *The update algorithm produces the same probability distribution for the clusters as a reclustering of the updated data using the original algorithm.*

To prove this, let us compute the occurrence probabilities of the successor

states – first the preexisting, then the newly added ones – and compare them to the frequency distribution of the states using the original process. For this proof, the following notations will be used: m is the number of walks arriving over the predecessor edge in question, $n = |T_{ij}|$ is the number of possible successors before and $n' = |T'_{ij}|$ the number of possible successors after the update. $N(e)$ is the number of walks choosing edge e as successor before the update, $N'(e)$ marks the same number after the update of the walks. As can easily be seen, $N(e)$ is binomially distributed with a $B(m, \frac{1}{n})$ distribution before the update due to the construction of the walks: It is the result of a repeated Bernoulli trial, a choice between the edge in question – which has a probability of $\frac{1}{n}$ of being chosen – and the other $n-1$ edges that share a total probability of $1 - \frac{1}{n} = \frac{n-1}{n}$. The choice is repeated m times, once for every walk arriving via the predecessor edge. The probability mass functions of the binomial distribution are denoted by $P(\cdot)$ and $P'(\cdot)$ respectively.

Theorem 4.1.2 *The frequency distribution of each member of T'_{ij}, i.e. after the appearance of new walks is $B(m, \frac{1}{n'})$.*

The proof is split into two parts. First, the correctness of the distribution for members of T_{ij} is shown, then the proof is made for the new edges, i.e. the members of $T'_{ij} \backslash T_{ij}$.

To this end, consider a successor e that was admissible before the change. The number of walks crossing it is $B(B(m, \frac{1}{n}), \frac{n}{n'})$-distributed: The probability of k walks containing e in this place after the update is composed of the probability that e was chosen i times ($i \geq k$) in the original process – this is the $B(m, \frac{1}{n})$ part – and that out of these i walks, k were not changed – this is where the term $\frac{n}{n'}$ comes from, since this is the probability of the walk not being truncated. The number of walks being preserved is thus $B(N(e), \frac{n}{n'})$-distributed.

The probability for k walks visiting e after the update is

$$P(N'(e) \quad = \quad k) = \sum_{i=k}^{m} \underbrace{P(N(e) = i)}_{i \text{ walks before update}} \underbrace{\left(\frac{n}{n'}\right)^{k} \left(\frac{n'-n}{n'}\right)^{i-k} \binom{i}{k}}_{i-k \text{ walks redistributed}}$$

consisting of the probability for i visits in the original walks and the binomial probability that k out of these are preserved. Since the original distribution of the visits is $B(m, \frac{1}{n})$, we obtain

$$= \sum_{i=k}^{m} \left(\frac{1}{n}\right)^{i} \left(\frac{n-1}{n}\right)^{m-i} \binom{m}{i} \left(\frac{n}{n'}\right)^{k} \left(\frac{n'-n}{n'}\right)^{i-k} \binom{i}{k}$$

writing out the two binomial coefficients and taking together the factor n in various numerators and denominators leads directly to

$$= \sum_{i=k}^{m} \left(\frac{1}{n}\right)^{m-k} (n-1)^{m-i} \frac{m!}{(m-i)!i!} \left(\frac{1}{n'}\right)^{k} \left(\frac{n'-n}{n'}\right)^{i-k} \frac{i!}{(i-k)!k!}$$

The factors $\left(\frac{1}{n}\right)^{m-k}$ and $\left(\frac{1}{n'}\right)^{k}$ are extracted from the sum, equally $m!$ and $k!$ from the two binomial coefficients; $i!$ cancels. Furthermore, the fraction is expanded by $(m-k)!$.

$$= \left(\frac{1}{n}\right)^{m-k} \left(\frac{1}{n'}\right)^{k} \frac{m!}{(m-k)!k!} \sum_{i=k}^{m} (n-1)^{m-i} \frac{(m-k)!}{(m-i)!(i-k)!} \left(\frac{n'-n}{n'}\right)^{i-k}$$

By transforming the sum's counter variable to start counting from 0, we obtain

$$= \left(\frac{1}{n}\right)^{m-k} \left(\frac{1}{n'}\right)^{k} \frac{m!}{(m-k)!k!} \sum_{i=0}^{m-k} (n-1)^{m-k-i} \frac{(m-k)!}{(m-k-i)!(i)!} \left(\frac{n'-n}{n'}\right)^{i}$$

The first three terms remain unchanged. The sum is the binomial representation of $\left((n-1) + (\frac{n'-n}{n'})\right)^{m-k}$, its first part is expanded by n' in order to obtain a common denominator:

$$= \left(\frac{1}{n}\right)^{m-k} \left(\frac{1}{n'}\right)^{k} \frac{m!}{(m-k)!k!} \left(\frac{nn'-n'+n'-n}{n'}\right)^{m-k}$$

The two middle n' in the last term cancel, leaving $\left(\frac{nn'-n}{n'}\right)^{m-k}$. By extracting the factor n^{m-k} from this term, the first term is canceled. The third term is the binomial coefficient which in total leads to

$$= \left(\frac{1}{n'}\right)^{k} \left(\frac{n'-1}{n'}\right)^{m-k} \begin{pmatrix} m \\ k \end{pmatrix} \qquad \square$$

which is exactly the probability of a $B(m, \frac{1}{n'})$-distributed variable having value k, thus proving the claim. This is true for any positive number of new successors.

The second part of the proof concerns the newly arrived successors. It uses the distribution of the number of visits to the preexisting successors given above. In order to give one proof for any number $l = n' - n$ of new successors, let us consider the total number of walks that use one of these new paths. Since the distribution of the number of walks is per construction identical on all new paths, this should lead to the l-time convolution of a $B(m, \frac{1}{n'})$ distribution. The following idea is used to compute the probability that the l new successors receive

k walks: If the total number of walks departing from the predecessor edge is m and all new paths together should receive k walks, the old paths must be visited by a total of $m-k$ walks. These are distributed over the preexisting paths in such a way that the successor edge $e_1 \in T_{ij}$ obtains between zero and $m-k$ walks with the probability distribution $B(m, \frac{1}{n'})$ as shown in the last proof, the successor edge $e_2 \in T_{ij}$ between zero and $m-k-N'(e_1)$ and so on. The n-th existing successor is visited by as many walks as are necessary in order to obtain a sum of $m-k$, in other words $m-k-\sum_{j=1}^{n-1} N'(e_j)$.

Thus we obtain as probability for the event of k walks selecting one of the new successors:

$$
\begin{aligned}
&P'(\sum_{j=n+1}^{n'} N'(e_j) = k) \\
&= \sum_{i_1=0}^{m-k} \sum_{i_2=0}^{m-k-i_1} \cdots \sum_{i_{n-1}=0}^{m-k-\sum_{j=1}^{n-2} i_j} \underbrace{P'(N'(e_1) = i_1)}_{e_1 \text{ has } i_1 \text{ walks}} \\
&\qquad \Pi_{j=2}^{n-1} \underbrace{P'(N'(e_j) = i_j | N'(e_1) = i_1, \ldots, N'(e_{j-1}) = i_{j-1})}_{e_j \text{ has } i_j \text{ walks, given } i_1, \ldots, i_{j-1}} \\
&\qquad \underbrace{P'(N'(e_n) = m-k-\sum_{j=1}^{n-1} i_j | N'(e_1) = i_1, \ldots, N'(e_{n-1}) = i_{n-1})}_{e_n \text{ receives the necessary walks to complete } m-k}
\end{aligned}
$$

In order for the l new successors to obtain k walks, the n old successors should together receive $m-k$ walks. The first of the old successors may obtain any number between 0 and $m-k$, the second one between 0 and what the first left of the $m-k$ walks and so on, until the last old successor is visited by as many walks as necessary to reach a sum of $m-k$. The number of walks for old successors follows a binomial distribution as we have seen in the last proof. The conditional probabilities here simply mean that $m-k$, the number of possible walks is reduced by the walks already assigned to one of the prior old successors. Writing these out, we obtain

$$
\begin{aligned}
&= \sum_{i_1=0}^{m-k} \cdots \sum_{i_{n-1}=0}^{m-k-\sum_{j=1}^{n-2} i_j} \left(\frac{1}{n'}\right)^{i_1} \left(\frac{n'-1}{n'}\right)^{m-i_1} \begin{pmatrix} m \\ i_1 \end{pmatrix} \\
&\quad \left(\frac{1}{n'-1}\right)^{i_2} \left(\frac{n'-2}{n'-1}\right)^{m-i_1-i_2} \begin{pmatrix} m-i_1 \\ i_2 \end{pmatrix} \\
&\quad \cdots \left(\frac{1}{n'-n+2}\right)^{i_{n-1}} \left(\frac{n'-n+1}{n'-n+2}\right)^{m-\sum_{j=1}^{n-1} i_j} \begin{pmatrix} m-\sum_{j=1}^{n-2} i_j \\ i_{n-1} \end{pmatrix} \\
&\quad \left(\frac{1}{n'-n+1}\right)^{m-k-\sum_{j=1}^{n-1} i_j} \left(\frac{n'-n}{n'-n+1}\right)^{m-\sum_{j=1}^{n-1} i_j - m + k + \sum_{j=1}^{n-1} i_j} \\
&\quad \begin{pmatrix} m-\sum_{j=1}^{n-1} i_j \\ m-k-\sum_{j=1}^{n-1} i_j \end{pmatrix}
\end{aligned}
$$

The first line contains the term for the first, the second for the second old successor. The second successor, given i_1 visits to e_1, has a $B(m-i_1, \frac{1}{n'-1})$ distribution. This continues till the $n-1$st one in the third line. The probability for the n-th old successor absorbing the rest of the $m-k$ walks is given in the last two lines. This expression can be simplified: The numerators from the second term in each line cancels the denominator of the first two terms of the following line. Equally, a part of the denominator from the binomial coefficient cancels the numerator of the coefficient in the next line – in the first line, for instance, this is $m-i_1$ from the term $\frac{m!}{(m-i_1)!i_1!}$. What remains is the denominator $\left(\frac{1}{n'}\right)^m$ from the first term, the numerator $(n'-n)^k$ from the last term, the $m!$ from the first binomial coefficient and the product $i_1!\ldots i_{n-1}!k!(m-k-\sum_{j=1}^{n-1} i_j)!$ remaining from the binomial coefficients' denominators.

$$
= \left(\frac{1}{n'}\right)^m (n'-n)^k \sum_{i_1=0}^{m-k} \cdots \sum_{i_{n-1}=0}^{m-k-\sum_{j=1}^{n-2} i_j} \frac{m!}{i_1!\ldots i_{n-1}!k!(m-k-\sum_{j=1}^{n-1} i_j)!}
$$

We expand by $(m-k-\sum_{j=1}^{n-2} i_j)!$, extract $\frac{m!}{k!}$, isolate the last sum and insert the factor $1^{i_{n-1}} 1^{m-k-\sum_{j=1}^{n-1} i_j}$ in order to obtain a binomial representation of $(1+1)^{m-k-\sum_{j=1}^{n-2} i_j} = 2^{m-k-\sum_{j=1}^{n-2} i_j}$:

$$
\begin{aligned}
&= \left(\frac{1}{n'}\right)^m (n'-n)^k \frac{m!}{k!} \sum_{i_1=0}^{m-k} \cdots \sum_{i_{n-1}=0}^{m-k-\sum_{j=1}^{n-3} i_j} \frac{1}{i_1!\ldots i_{n-2}!(m-k-\sum_{j=1}^{n-2} i_j)!} \\
&\quad \sum_{i_{n-1}=0}^{m-k-\sum_{j=1}^{n-2} i_j} 1^{i_{n-1}} 1^{m-k-\sum_{j=1}^{n-1} i_j} \frac{(m-k-\sum_{j=1}^{n-2} i_j)!}{i_{n-1}!(m-k-\sum_{j=1}^{n-1} i_j)!}
\end{aligned}
$$

We add another factor 1 in order to obtain the binomial representation of $3^{m-k-\sum_{j=1}^{n-3} i_j}$:

$$= \left(\frac{1}{n'}\right)^m (n'-n)^k \frac{m!}{k!} \sum_{i_1=0}^{m-k} \dots \sum_{i_{n-1}=0}^{m-k-\sum_{j=1}^{n-3} i_j} \frac{1}{i_1! \dots i_{n-2}! (m-k-\sum_{j=1}^{n-2} i_j)!} 2^{m-k-\sum_{j=1}^{n-2} i_j} 1^{i_{n-2}}$$

We continue calculating the binomial representation of $(2+1), (3+1), \dots, (n-2)$ to the respective power, until we obtain

$$= \left(\frac{1}{n'}\right)^m (n'-n)^k \frac{m!}{k!(m-k)!} \sum_{i_1=0}^{m-k} \frac{(m-k)!}{i_1!(m-k-i_1)!} (n-1)^{m-k-i_1} 1^{i_1}$$

Once again, the sum is the binomial representation of the $m-k$-th power of $((n-1)+1) = n$.

$$= \left(\frac{1}{n'}\right)^m (n'-n)^k \frac{m!}{k!(m-k)!} n^{m-k}$$

By substituting n by $n'-l$, we obtain l^k for the second and $(n'-l)^{m-k}$ for the fourth term. Finally, we split the factor $\left(\frac{1}{n'}\right)^m$ and distribute it over the second and fourth term in order to obtain

$$= \left(\frac{l}{n'}\right)^k \left(\frac{n'-l}{n'}\right)^{m-k} \binom{m}{k} \qquad \square$$

which proves the assumption for the sum of the number of walks visiting any one of the new successors.

The last line gives the probability for the sum of l identically $B(m, \frac{1}{n'})$ distributed variables having value k. Since the number of walks at the new successors is indeed identically distributed, it follows that the number of walks visiting each of them is $B(m, \frac{1}{n'})$ distributed.

This concludes the proof for case (1): The frequency distributions – for nothing else is given by $P'(N'(e) = k)$ – for preexisting and for new successors are identical for both the updated walks and for the original walks when executed on the updated similarity matrix. The update does not affect any other probabilities; thus the two processes are equivalent.

4.1.2 Illegal Successors

Case (2), on the other hand, describes the scenario where a successor edge becomes illegal after the update. As a consequence, all walks crossing that edge have to be modified since a transition to this edge is no longer possible. For every walk concerned, the part of the walk beginning with that edge must be deleted from the walk data base and a new (partial) walk has to be started departing from the predecessor state in the same manner as in the initial walk. In other words, a successor is chosen with equal probability from the remaining $|T'_{ij}|$ edges.

Formally, this implies the following algorithm for an illegal successor e:

For all walks $(\ldots, e', e, \ldots)$:
  Truncate the walk, starting from e
  Pick e'' from a uniform distribution over $T'_{e'}$
  Insert e'' at the end of the walk
  Continue the walk by picking a successor from $T'_{e''}$
  Follow the original algorithm for the rest of the walk
End

In our example (cf. page 70), the walk data base contains the walks $DFBCG$, EGC, BGE, ACG, BCA, $CBACG$, GBF, and $DACG$ after the first update. Due to the second update, the walk $DFBCG$ is now invalid and must be reconsidered. After the first two steps DFB, we have the new set of successors $T'_{FB} = \{BA\}$, which finally leads to the walk $DFBACG$. For the component cluster method (cf. page 72), we have a new level 5 after the update that contains the cluster $\{C, G\}$ and the other objects as singletons. Furthermore, the node A joins the cluster $\{C, G\}$ at level 4. The final clustering is given in Tab. 4.1.

When considering the walk context clusters, the nodes A and B are once again part of each others' cluster at level 0.5 after the second update. On the other hand, D is removed from the clusters for B and F and vice versa at level 0.2 since the first step of the walk $DFBACG$ has only a level $l^+ = \frac{1}{6}$, and F and A are added to each other's cluster at level 0.2 as can be seen in Tab. 4.2.

Table 4.1: Component clusters for the example after the second update

level	clusters
1	$\{A, B, C, D, E, F, G\}$
2	$\{A, B, C, E, F, G\}, \{D\}$
3	$\{A, B, C, G\}, \{D\}, \{E\}, \{F\}$
4	$\{\mathbf{A}, \mathbf{C}, \mathbf{G}\}, \{B\}, \{D\}, \{E\}, \{F\}$
5	$\{\mathbf{A}\}, \{\mathbf{B}\}, \{\mathbf{C}, \mathbf{G}\}, \{\mathbf{D}\}, \{\mathbf{E}\}, \{\mathbf{F}\}$

Table 4.2: Walk context clusters for the example after the second update

l^+	A	B	C	D
0.2	$\{B, C, D, \mathbf{F}, G\}$	$\{A, C, E, F, G\}$	$\{A, B, D, E, F, G\}$	$\{A, C, G\}$
0.5	$\{\mathbf{B}, C, G\}$	$\{\mathbf{A}, C, G, F\}$	$\{A, B, G\}$	$\{\}$
0.8	$\{\}$	$\{\}$	$\{G\}$	$\{\}$

l^+	E	F	G
0.2	$\{B, C, G\}$	$\{\mathbf{A}, B, C, G\}$	$\{A, B, C, D, E, F\}$
0.5	$\{G\}$	$\{B\}$	$\{A, B, C, E\}$
0.8	$\{\}$	$\{\}$	$\{C\}$

Of course, the update procedure for illegal successors also produces the same result as the original algorithm on the updated data set.

Theorem 4.1.3 *The frequency distribution of each member of T'_{ij}, i.e. after paths have become illegal is $B(m, \frac{1}{n'})$.*

For the proof that this method yields the same distribution as if the walks had been executed from scratch again, consider the case where $n' = n - l$, i.e. l paths are no longer possible. Without loss of generality, let $e_1, \ldots, e_l$ be the edges that are no longer in T'_{ij}. The respective number of walks choosing one of these edges as a successor for the updated edge is given as $N(e_1), \ldots, N(e_l)$. After the update, the distribution of the number of walks selecting a certain successor $e \in T'_{ij}$ should follow a $B(m, \frac{1}{n'})$ distribution, i.e. the probability of k walks out of m selecting a given successor state e as successor for the updated edge among the n' admissible ones should be

$$P(N'(e) = k) = (\frac{1}{n'})^k (\frac{n' - 1}{n'})^{m-k} \binom{m}{k}$$

The probability distribution of the number of walks using successor e after the update is the number of walks that crossed that edge before the update plus the number of walks that are added by the update. The first is established by the $B(m, \frac{1}{n})$ distribution for the original process. The latter is computed from the l-time convolution of the $B(m, \frac{1}{n})$ distribution for the now illegal successors combined with the probability that their walks are redistributed to the given successor e.

The first term gives the probability of the successor in question already having h walks. The second term gives the probability that i walks contain one of the l now illegal successors and that $k - h$ out of these are attributed to the path in question.

$$
\begin{aligned}
&P'(N(e)=k)\\
&=\sum_{h=0}^{k} \underbrace{P(N(e)=h)}_{h \text{ walks on } e \text{ before update}} \sum_{i=k-h}^{m-h} \underbrace{P(\sum_{j=1}^{l} N(e_j)=i | N(e)=h)}_{i \text{ walks to be deleted}}\\
&\underbrace{\left(\frac{1}{n'}\right)^{k-h}\left(\frac{n'-1}{n'}\right)^{i-k+h}\left(\begin{array}{c} i \\ k-h \end{array}\right)}_{k-h \text{ deleted walks redistributed to } e}
\end{aligned}
$$

The first probability is, as discussed, a binomial distribution with m realizations. The second probability is the l-time convolution of binomial distributions, and denotes the probability that, given h walks visiting e, i of the remaining $m-h$ walks are candidates for redistribution. Inserting the concrete probabilities leads to

$$
\begin{aligned}
=&\sum_{h=0}^{k}\left(\frac{1}{n}\right)^{h}\left(\frac{n-1}{n}\right)^{m-h}\left(\begin{array}{c} m \\ h \end{array}\right)\\
&\sum_{i=k-h}^{m-h}\left(\frac{l}{n-1}\right)^{i}\left(\frac{n-l-1}{n-1}\right)^{m-h-i}\left(\begin{array}{c} m-h \\ i \end{array}\right)\\
&\left(\frac{1}{n'}\right)^{k-h}\left(\frac{n'-1}{n'}\right)^{i-k+h}\left(\begin{array}{c} i \\ k-h \end{array}\right)
\end{aligned}
$$

Writing out the binomial coefficients and summarizing factors in each line gives us

$$
\begin{aligned}
=&\left(\frac{1}{n}\right)^{m} m!\sum_{h=0}^{k}(n-1)^{m-h}\frac{1}{(m-h)!h!}\\
&\left(\frac{1}{n-1}\right)^{m-h}\sum_{i=k-h}^{m-h} l^{i}(n-l-1)^{m-h-i}\frac{(m-h)!}{i!(m-h-i)!}\\
&\left(\frac{1}{n'}\right)^{i}(n'-1)^{i-k+h}\frac{i!}{(k-h)!(k-h-i)!}
\end{aligned}
$$

$\left(\frac{1}{n-1}\right)^{m-h}$, $(m-h)!$, and $i!$ cancel. $n-l$ can be rewritten as n', which allows us to simplify $(n-l-1)^{m-h-i}(n'-1)^{i-k+h}$ to $(n'-1)^{m-k}$.

$$
=\left(\frac{1}{n}\right)^{m} m!(n'-1)^{m-k}\sum_{h=0}^{k}\frac{1}{(k-h)!h!}\sum_{i=k-h}^{m-h}\left(\frac{l}{n'}\right)^{i}\frac{1}{(k-h-i)!(m-h-i)!}
$$

We substitute the counter variable i to start counting from zero, expand by $(m-k)!$, and extract $(\frac{l}{n'})^{k-h}$ from the last sum. Furthermore, we multiply with $1^{m-k-i} = 1$:

$$= \left(\frac{1}{n}\right)^m \frac{m!}{(m-k)!}(n'-1)^{m-k} \sum_{h=0}^{k} \frac{1}{(k-h)!h!} \left(\frac{l}{n'}\right)^{k-h} \sum_{i=0}^{m-k} 1^{m-k-i} \left(\frac{l}{n'}\right)^i \frac{(m-k)!}{i!(m-k-i)!}$$

With this, we obtain the last sum as binomial representation of $\left(1+\frac{l}{n'}\right)^{m-k}$:

$$= \left(\frac{1}{n}\right)^m \frac{m!}{(m-k)!}(n'-1)^{m-k} \sum_{h=0}^{k} \frac{1}{(k-h)!h!} \left(\frac{l}{n'}\right)^{k-h} \left(1+\frac{l}{n'}\right)^{m-k}$$

Unifying the denominator and rewriting $n = n' + l$ leads to:

$$= \left(\frac{1}{n'+l}\right)^m \frac{m!}{(m-k)!}(n'-1)^{m-k} \sum_{h=0}^{k} \frac{1}{(k-h)!h!} \left(\frac{l}{n'}\right)^{k-h} \left(\frac{n'+l}{n'}\right)^{m-k}$$

We summarize the terms containing $n'+l$ and n', multiply by $1^h = 1$, and expand by $k!$ to obtain the binomial representation of $(1+\frac{l}{n'})^k$:

$$= \left(\frac{1}{n'+l}\right)^k \left(\frac{n'-1}{n'}\right)^{m-k} \frac{m!}{(m-k)!k!} \sum_{h=0}^{k} \frac{k!}{(k-h)!h!} \left(\frac{l}{n'}\right)^{k-h} 1^h$$

Rewriting the binomial coefficients and unifying the denominators in the last term gives:

$$= \left(\frac{1}{n+l}\right)^k \left(\frac{n'-1}{n'}\right)^{m-k} \binom{m}{k} \left(\frac{n'+l}{n'}\right)^k$$

which is then simplified to

$$= \left(\frac{1}{n'}\right)^k \left(\frac{n'-1}{n'}\right)^{m-k} \binom{m}{k} \qquad \square$$

The resulting distribution is once more a $B(m, \frac{1}{n'})$ distribution which proves the equivalence of the transition probabilities and thus the equivalence of the two processes for case (2).

4.1.3 New Paths and Illegal Successors

Up to now, we have considered the two cases and separately. But it may happen that at a node, both one or more new paths are opened while some existing paths become invalid due to changes on just one of the edges – we will treat the case of multiple edge updates below, in section 4.2. However, a walk can never be affected by both cases at once: If, in our example, k is incremented from two to four, the sequence GBF becomes possible and the sequence FBC is now illegal. As can be seen the edge BF is part of both paths, but the paths traverse it in different directions. Since per construction, an edge cannot be crossed twice in the same walk due to the strictly increasing similarity restriction, the two sequences cannot be part of the same walk. This is equivalent to the assertion in Fig. 3.5 that G' is cycle-free. Consequently, the two events do not influence each other and thus can be treated sequentially, i.e. by pretending that they happened one after the other. As the updates do not deteriorate the cluster quality, the order of new paths emerging and illegal successors being removed is not important for the outcome.

4.1.4 New Nodes

The introduction of a new node into the graph triggers two actions: First, the appropriate number of walks starting from the new node must be executed using the original algorithm, and second, the walks containing the new node's neighbors must be scrutinized whether the appearance of the new edges has opened new possible paths as described in case (1) above. The first action is straightforward and does not differ from the initial walks described in section 3.1.1, and the second one can be reduced to case (1).

Evidently, the insertion of new nodes does not invalidate any old walks.

4.1.5 Deletion of Nodes

The deletion of the node is the inverse of the case of new nodes: First, all walks starting from the deleted node must be removed from the walk data base. Afterwards, the walks visiting the node in question must be pruned at its first occurrence and restarted from there as described for case (2).

The removal of a node does not cause the formation of new paths.

4.2 Concurrent Updates of the Similarity Matrix

Up to this point, we have treated the updates as separate events that each triggered its own update process. For updates arriving in batches, two possible paths of

action are conceivable. The first one is to only consider one update at a time, and to use the update procedure outlined above. This is certainly the most simple implementation of a batch update, yet it is well suited for batches where only few interdependencies between the changes inside the batch exist, i.e. if few walks are affected by more than one change. Furthermore, this update method is cheap to implement as it does not require additional data structures to account for the current state of the walk update process.

Alternatively, it is possible to supplement each entry in the similarity matrix with a delta field that contains the difference between the old and the new similarity assigned to this entry if there is a difference, or zero else. In that case, the update algorithm first fills the delta fields according to all updates in the batch. It then considers all object pairs affected by an update, taking the appropriate steps for each of them, but making the choices for successors based on the *updated* matrix. These walks are marked as updated along with the place where the truncation took place. If, in a later stage of the batch processing, the algorithm encounters such a walk in a step that is located after the truncation, the walk is excluded from the redistribution process of old walks to new paths since in its (re-)creation, the correct probabilities have already been used. The case of deleting the new walk because of a change after the truncation is not relevant here since per construction the updated walk would not have chosen a successor that has become illegal due to the update.

Obviously, the problem with this approach is the large overhead for the similarity matrix updates and for marking new walks. This makes the method only practicable for cases where a considerable number of updates with a high amount of interdependencies has to be carried out.

4.3 Complexity of the Update Procedure

Given the procedure for updating RRW clusters, it is of course very interesting to know the computational complexity of these updates. Let us consider the two basic cases of new paths and illegal successors. To this end, the following definitions shall be introduced: Let $\bar{d}$ be the average degree of a node in the similarity graph G. It follows that G has $\frac{\bar{d}n}{2}$ edges. Furthermore, we can state that each walk has a length of $O(\log n)$ as seen in section 3.2.1. In total, this means that, by starting k walks at each of the n nodes a total of $O(nk \log n)$ steps is executed. In other word, every node is visited on the average by $O(k \log n)$ walks, each edge by $O(\frac{2k \log n}{\bar{d}})$ walks. It is important to note that not every change in the raw baskets triggers an update, thus the complexity for updates will be in practice lower than the $O(\frac{k \log^2 n}{\bar{d}})$ derived here.

4.3.1 New Successors

When a new successor becomes admissible, an average of $O(k \log n)$ walks must be considered out of which $O(\frac{2k \log n}{d})$ walks must be chosen, truncated and partially recomputed using the newly arrived successor. Since a walk has a length of $O(\log n)$ and can be expected to be cut in half at this step, deleting it involves $O(\frac{1}{2} \log n)$ steps. If the walk data can be accessed in linear time e.g. using a hash table this leads to a total complexity of $O(\frac{k \log^2 n}{d})$ for the deletion of all walk parts. The same complexity applies to the insertion of the new walk fragments of average length $O(\frac{1}{2} \log n)$, thus the overall complexity for a new successor becoming possible is $O(\frac{k \log^2 n}{d})$.

4.3.2 Illegal Successors

The argumentation is comparable to the case above, only that the choice process selects a place to continue the truncated walks rather than the walks to be truncated and deletion and insertion are executed in other locations. Thus the overall complexity for dealing with an illegal successor is equally $O(\frac{k \log^2 n}{d})$. In total, this implies a slightly increased complexity for the walk stage of the clustering process. However, keeping in mind the relation between the object set with about 0.8 million active documents in the case of the OPAC data set and weekly updates of a few thousand documents, the update procedure still is largely favorable.

4.3.3 Cluster Construction

The complexity of the updates of the actual clusters depends on the cluster construction method chosen.

For walk context clusters, the clusters can be computed online when needed with complexity of $O(\log^2 n)$ as proved in section 3.2.1. It is thus not necessary to explicitly update the clusters since the requests following the update will automatically access the up-to-date data base.

For component clusters, each edge newly included in the data base must be scrutinized whether it connects two formerly unconnected clusters at any level. This is achieved in linear time to the number of available levels, since it suffices to check for each level whether the two nodes incident to the edge are in the same cluster. If this is the case, no action is needed; otherwise, the two clusters must be merged.

If, on the other hand, an edge is no longer included in the data base because of the update at a given number of levels, the algorithm must search for alternative paths in the H_k graph introduced in section 3.1.2. If such a path is found, the cluster is still connected at that level, otherwise, the two components, each containing

one of the nodes incident to the edge in question, form two separate clusters from that point on. By using a simple depth-first search, a complexity of $O(n\bar{d})$ is possible which is consistent with the complexity given in section 3.2.1, although here n is a rather coarse upper bound on the size of the cluster containing the deleted edge.

4.3.4 Simulation

In order to determine the real-world performance of the update procedure, a simulation was executed on an AMD Athlon64 X2 4200+ machine using the library data set described in section 3.3.1. This machine can handle about 2 walks per second when executing the original, non-incremental implementation on this data set. A set of 5000 updated edges was created by randomly selecting edges from the data base and increasing their similarity weight by an integer that is randomly picked from a uniform distribution over the interval [1,10].

The simulation shows that the current implementation of the update algorithm is able to handle about 0.5 updates per second. It follows that, on a data set comprising n nodes and having characteristics comparable to the library data set, both methods take about the same time for the computation of an up-to-date clustering if the number of updates equals $\frac{n}{4}$. If the number is smaller, then the update algorithm is faster; if it is larger, the original algorithm should be used to recluster the whole data set.

In practice, the additional possibility of real-time updates should be kept in mind. Of course, a real-time update is only possible when using the update algorithm: On a site with moderate traffic, it is possible to integrate changes occurring during operation online. This is a strong advantage compared to the original algorithm, where changes can only be computed offline and thus with a considerable delay. On the other hand, the more frequently the updates are integrated, the less the system can profit from aggregating several subsequent increases in the weight of the same edge or edges, which leads to a total increase of the computation time needed. As a result, the update frequency should be fixed individually based on the following factors:

- real-time requirements for the clusters that follow from the application at hand,
- the computational resources, and
- the update frequency of the similarity matrix.

4.4 Evaluation

The requirements that the update process for the restricted random walk cluster algorithm has to fulfill were outlined at the beginning of this chapter: To efficiently incorporate deletions, insertions and changes in the similarity matrix into the existing cluster structure while maintaining the probability distribution of the original algorithm such that it never becomes necessary to recluster the data, independent of the number of updates.

We have seen that the update procedure has a time complexity of $O(\frac{lk \log^2 n}{d})$ for l significant changes in the similarity matrix which allows an efficient handling of the update process. Equally, a reclustering is not necessary since the probability distributions of the updated walks – and thus also those of the updated clusters – are exactly the same as the ones that could be obtained by reclustering the changed data set using the original cluster algorithm.

This constitutes a considerable advance in comparison with the methods presented in chapter 2. First, many of the algorithms are only capable of adding new objects, e.g. gravitational clustering [CHO02] or similarity histogram clustering [HK03]. Some algorithms like incremental OPTICS [KKG03] or star clusters [APR97] are also capable of deleting objects, which would – normally under considerable computational expense – also allow the integration of changes in the similarity matrix by removing and reinserting the respective objects. Additionally, the clusters produced by these algorithms are often dependent on the order in which the object are presented. This effect does not occur with RRW clustering.

The algorithms from the data base domain, on the other hand, can often cope with changes, but define clusters rather awkwardly as groups of objects that are related – in most cases, in terms of their usage pattern – and fit in one memory page. Furthermore, many of these algorithms are greedy and thus order-dependent, and some approaches only offer criteria for determining the point in time when a complete reclustering is necessary [MK94].

Finally, the domain of mobile, wireless, and/or ad-hoc networks seems to be a promising application area for dynamic cluster algorithms, but the solutions proposed in this field often enough sacrifice cluster quality for low communication overhead and low computational costs. Additionally, clusters are often simply defined as the area around a node that is directly reachable via the wireless network without taking into account the spatial density distribution of the nodes inside the network [KVCP97].

To sum it up: Literature research did not reveal a single update algorithm that fulfills all of the requirements stipulated here. Each one of the algorithms cited above lacks at least one feature from the requirements list, whereas the algorithm developed in this thesis is capable of performing updates of the cluster structure while fulfilling all of the above requisites.

Chapter 5

Conclusion and Outlook

This thesis has presented a solution for the problem of keeping clusters up to date facing constantly changing data sets, both in size and internal structure. After laying some foundations for general clustering and for stochastic processes, existing solutions for the handling of dynamic data sets were reviewed. A large part only deals with growing and possibly shrinking data sets, thus ignoring the fact that objects inside the data set may change their respective positions. On the other hand, solutions for mobile, wireless ad hoc scenarios take changing distances into account. Unfortunately, the solutions discussed focus their attention rather on heuristics that allow clusters to be maintained with minimal computational and communicational effort, but at the cost of the clusters' quality. In the domain of data bases, algorithms were found that deal with the required kinds of changes, but that are also too restrained by the application, namely to organize the objects on a secondary storage, thus using a fixed cluster size.

Therefore, the restricted random walk algorithm presented in chapter 3 that has been successfully used in different scenarios was extended in such a way as to integrate the aforementioned changes with a minimum of computational complexity. Furthermore, it has been shown that the clusterings produced by the update algorithm have the same probability distribution as those generated by the original algorithm on the data set incorporating the changes. The computational cost of the update algorithm allows to efficiently compute these changes.

For the future, there remain further interesting research questions in the context of RRW clustering. The first complex is the question of the algorithm's behavior subject to differing numbers of walks. As was discussed in section 3.2.2, the number of walks started at each node has an important influence on the characteristics, especially the stability of the clusters. For component clusters and an infinite number of walks, an analysis technique has been sketched in this thesis that now must be extended both to walk context clusters and to finite numbers of walks. The final goal is to be able to give a number of walks in order to obtain

a desired property of the clusters – for instance a given stability with a minimum number of walks.

Furthermore, the walk process in itself should be more thoroughly scrutinized; for example, the question of the exact connection between random graphs and restricted random walks is yet unanswered.

Some applications do not have a symmetric similarity matrix. For these, it would be challenging to integrate directional aspects into the algorithm and to find an interpretation for the resulting clusters.

Another open question concerns the use of restricted random walks for the detection of bridge elements that do not belong to clusters but lie between several clusters. The characteristics of such bridge elements have been described in section 3.1.2.

In the application area, further scenarios for the deployment of the algorithm are in preparation. A promising use for RRW clustering might be in the domain of collaborative search: Search terms are clustered based on past queries, and when one of these terms is entered in an ongoing search, the system is able to recommend terms that complement the given one. These recommendations can serve both to broaden or to narrow down the search, depending on the exact construction of the similarity matrix.

Appendix A

A Sample Raw Basket

10029087§BLB_OPAC := {1}
10218646§BLB_OPAC := {2}
1048708§BLB_OPAC := {1}
10754692§UBKA_OPAC := {4}
10823891§UBKA_OPAC := {4}
1126473§BLB_OPAC := {1}
1191048§BLB_OPAC := {3}
1195261§BLB_OPAC := {1}
1199273§BLB_OPAC := {1}
1255566§BLB_OPAC := {1}
1383380§BLB_OPAC := {1}
15442§BLB_OPAC := {1}
158515§BLB_OPAC := {1}
1586739§BLB_OPAC := {1}
1613121§BLB_OPAC := {12}
1613121§UBKA_OPAC := {4}
1721232§BLB_OPAC := {1}
1808764§BLB_OPAC := {1}
1878727§BLB_OPAC := {2}
1878749§BLB_OPAC := {1}
1917029§BLB_OPAC := {1}
1921827§BLB_OPAC := {1}
193480§BLB_OPAC := {1}
1984206§BLB_OPAC := {3}
2079804§BLB_OPAC := {1}
2093415§BLB_OPAC := {1}
2172028§BLB_OPAC := {1}
2303507§BLB_OPAC := {5}

2361281§BLB_OPAC := {1}
2541879§BLB_OPAC := {1}
2582950§BLB_OPAC := {1}
2582958§BLB_OPAC := {1}
2582963§BLB_OPAC := {1}
2600767§BLB_OPAC := {1}
2710947§BLB_OPAC := {1}
2711858§BLB_OPAC := {1}
2743760§BLB_OPAC := {1}
283203§BLB_OPAC := {1}
2967256§BLB_OPAC := {1}
3070551§BLB_OPAC := {1}
321630§BLB_OPAC := {1}
3483081§BLB_OPAC := {1}
348801§BLB_OPAC := {1}
3533485§BLB_OPAC := {1}
388487§BLB_OPAC := {1}
3984289§BLB_OPAC := {1}
4008295§BLB_OPAC := {3}
4038156§BLB_OPAC := {1}
405013§BLB_OPAC := {1}
4065003§BLB_OPAC := {1}
4383274§BLB_OPAC := {1}
450036§BLB_OPAC := {1}
4720977§BLB_OPAC := {1}
4748244§BLB_OPAC := {12}
4763375§BLB_OPAC := {1}
4910341§BLB_OPAC := {1}
4942781§BLB_OPAC := {1}
5006968§BLB_OPAC := {1}
523894§BLB_OPAC := {1}
5294064§BLB_OPAC := {1}
5349248§BLB_OPAC := {1}
5385116§BLB_OPAC := {1}
5615297§BLB_OPAC := {1}
5706587§BLB_OPAC := {1}
573865§BLB_OPAC := {5}
573865§UBKA_OPAC := {4}
6086536§BLB_OPAC := {1}
6113137§BLB_OPAC := {1}
6113147§BLB_OPAC := {1}

6138003§BLB_OPAC := {1}
6174272§BLB_OPAC := {2}
619188§BLB_OPAC := {1}
627927§BLB_OPAC := {1}
633383§BLB_OPAC := {1}
6344176§BLB_OPAC := {1}
6364441§BLB_OPAC := {1}
650759§BLB_OPAC := {1}
6517614§UBKA_OPAC := {4}
6573434§BLB_OPAC := {4}
6624916§BLB_OPAC := {4}
6774232§BLB_OPAC := {3}
7062354§BLB_OPAC := {1}
7457661§BLB_OPAC := {1}
7588657§BLB_OPAC := {7}
768799§BLB_OPAC := {1}
7691438§BLB_OPAC := {1}
7816891§BLB_OPAC := {1}
7816930§BLB_OPAC := {1}
786247§BLB_OPAC := {1}
792405§BLB_OPAC := {1}
8010581§BLB_OPAC := {1}
8150443§BLB_OPAC := {2}
8466829§BLB_OPAC := {1}
8545961§BLB_OPAC := {1}
861110§BLB_OPAC := {1}
8833645§BLB_OPAC := {1}
8875811§BLB_OPAC := {1}
8948020§BLB_OPAC := {1}
9034741§BLB_OPAC := {4}
9034776§BLB_OPAC := {3}
909767§BLB_OPAC := {1}
9198669§BLB_OPAC := {1}
9733275§UBKA_OPAC := {4}
9803654§BLB_OPAC := {1}
9831246§BLB_OPAC := {1}
9961261§BLB_OPAC := {4}
ID := {9034706§BLB_OPAC}

Appendix B

The Deep South Raw Data

Table B.1: The original Deep South data set by Davis et al. [DGG48]

ids	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	x	x	x	x	x	x	-	x	x	-	-	-	-	-
2	x	x	x	-	x	x	x	x	-	-	-	-	-	-
3	-	x	x	x	x	x	x	x	x	-	-	-	-	-
4	x	-	x	x	x	x	x	x	-	-	-	-	-	-
5	-	-	x	x	x	-	x	-	-	-	-	-	-	-
6	-	-	x	-	x	-	-	x	-	-	-	-	-	-
7	-	-	-	-	x	x	x	x	-	-	-	-	-	-
8	-	-	-	-	-	x	-	x	x	-	-	-	-	-
9	-	-	-	-	x	-	x	x	x	-	-	-	-	-
10	-	-	-	-	-	-	x	x	x	-	-	x	-	-
11	-	-	-	-	-	-	-	x	x	x	-	x	-	-
12	-	-	-	-	-	-	-	x	x	x	-	x	x	x
13	-	-	-	-	-	-	x	x	x	x	-	x	x	x
14	-	-	-	-	-	x	x	-	x	x	x	x	x	x
15	-	-	-	-	-	-	x	x	-	x	x	x	-	-
16	-	-	-	-	-	-	-	x	x	-	-	-	-	-
17	-	-	-	-	-	-	-	-	x	-	x	-	-	-
18	-	-	-	-	-	-	-	-	x	-	x	-	-	-

List of Figures

List of Tables

Index

Bibliography

[AABV99] Pankaj K. Agarwal, Lars Arge, Gerth Stølting Brodal, and Jeffrey Scott Vitter. I/O-efficient dynamic point location in monotone planar subdivisions. In Robert E. Tarjan and Tandy Warnow, editors, *SODA '99: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 11–20, Philadelphia, 1999. SIAM.

[Ada82] Douglas Adams. *Life, the Universe and Everything*. Pan Books, London, 1982.

[AHWY03] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer, editors, *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases*, pages 81–92, San Francisco, 2003. Morgan Kaufmann.

[AKCM90] Stanley C. Ahalt, Ashok K. Krishnamurthy, Prakoon Chen, and Douglas E. Melton. Competitive learning algorithms for vector quantization. *Neural Networks*, 3(3):277 – 290, 1990.

[AM70] J. Gary Augustson and Jack Minker. An analysis of some graph theoretical cluster techniques. *Journal of the ACM*, 17(4):571 – 588, 1970.

[APR97] Javed Aslam, Katya Pelekhov, and Daniela Rus. Computing dense clusters on-line for information organization. Technical Report PCS-TR97–324, Department of Computer Science, Dartmouth, 1997.

[APR98] Javed Aslam, Katya Pelekhov, and Daniela Rus. Static and dynamic information organization with star clusters. In Georges

Gardarin, James C. French, Niki Pissinou, Kia Makki, and Luc Bouganim, editors, *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management*, pages 208 – 217, New York, 1998. ACM Press.

[APR99] Javed Aslam, Katya Pelekhov, and Daniela Rus. A practical clustering algorithm for static and dynamic information organization. In Robert E. Tarjan and Tandy Warnow, editors, *SODA '99: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 51 – 60, Philadelphia, 1999. SIAM.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[Bac00] Louis Bachelier. Théorie de la spéculation. *Annales Scientifiques de l'École Normale Supérieure, Série 3*, 17:21–86, 1900.

[Bar02] Daniel Barbará. Requirements for clustering data streams. *SIGKDD Explorations Newsletter*, 3(2):23 – 27, 2002.

[Bas99] Stefano Basagni. Distributed clustering for ad hoc networks. In *Proceedings of the Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN '99)*, pages 310–315, Piscataway, 1999. IEEE Press.

[BC01] Daniel Barbará and Ping Chen. Tracking clusters in evolving data sets. In Ingrid Russell and John F. Kolen, editors, *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, pages 239–243, Menlo Park, 2001. AAAI Press.

[Bez81] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

[BKKG88] Jay Banerjee, Won Kim, Sung-Jo Kim, and Jorge F. Garza. Clustering a DAG for CAD databases. *IEEE Transactions on Software Engineering*, 14(11):1684–1699, 1988.

[BLKR98] Al Borchers, Dave Leppik, Joseph Konstan, and John Riedl. Partitioning in recommender systems. Technical Report 98–023, University of Minnesota, Minneapolis, 1998.

[BMSM90] David A. Bell, F. J. McErlean, P. M. Stewart, and Sally I. McClean. Application of simulated annealing to clustering tuples in databases. *Journal of the American Society for Information Science: JASIS*, 41(2):98–110, 1990.

[BN70] Michael N. Barber and Barry W. Ninham. *Random and Restricted Walks: Theory and Applications*. Gordon and Breach, New York, 1970.

[Boc74] Hans Hermann Bock. *Automatische Klassifikation*. Vandenhoeck&Ruprecht, Göttingen, 1974.

[Bol01] Bela Bollobas. *Random Graphs*. Cambridge University Press, Cambridge, 2nd edition, 2001.

[BS96] Frédérique Bullat and Michel Schneider. Dynamic clustering in object databases exploiting effective use of relationships between objects. In Pierre Cointe, editor, *ECCOP'96 – Object-Oriented Programming, 10th European Conference*, pages 344–365, Heidelberg, 1996. Springer.

[Can93] Fazli Can. Incremental clustering for dynamic information processing. *ACM Transactions on Information Systems*, 11(2):143 – 164, 1993.

[CCFM97] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In F. Tom Leighton and Peter Shor, editors, *STOC '97: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 626–635, New York, 1997. ACM Press.

[CD90] Fazli Can and Nicklas D. II Drochak. Incremental clustering for dynamic document databases. In *Proceedings of the 1990 Symposium on Applied Computing*, pages 61–67, Washington, 1990. IEEE Computer Society.

[CEQZ06] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM Conference on Data Mining*, pages 326–337, Philadelphia, 2006. SIAM.

[CFSF95] Fazli Can, Edward A. Fox, Cory D. Snavely, and Robert K. France. Incremental clustering for very large document databases: Initial MARIAN experience. *Information Sciences – Informatics and*

Computer Science: An International Journal, 84(1–2):101 – 114, 1995.

[Cha94] Bidyut Baran Chaudhuri. Dynamic clustering for time incremental data. *Pattern Recognition Letters*, 15(1):27–34, 1994.

[Cha02] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thirty-fourth Annual ACM Symposium on Theory of Computing*, pages 380–388, New York, 2002. ACM Press.

[CHK91] Jason Cong, Lars Hagen, and Andrew B. Kahng. Random walks for circuit clustering. In *Proceedings of the Fourth Annual ASIC Conference and Exhibit*, pages 14.2.1–14.2.4, Washington, 1991. IEEE Computer Society.

[CHO02] Chien-Yu Chen, Shien-Ching Hwang, and Yen-Jen Oyang. An incremental hierarchical data clustering algorithm based on gravity theory. In M.S. Chen, P.S. Yu, and B. Liu, editors, *Advances in Knowledge Discovery and Data Mining: 6th Pacific-Asia Conference, PAKDD 2002*, pages 237–250, Heidelberg, 2002. Springer.

[CHO05] Chien-Yu Chen, Shien-Ching Hwang, and Yen-Jen Oyang. A statistics-based approach to control the quality of subclusters in incremental gravitational clustering. *Pattern Recognition*, 38(12):2256–2269, 2005.

[CKT06] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Proceedings of the 2006 ACM SIGKDD*, New York, 2006. ACM Press.

[CM65] David R. Cox and H. D. Miller. *The Theory of Stochastic Processes*. Chapman & Hall, London, 1965.

[CO87] Fazli Can and Esen A. Ozkarahan. A dynamic cluster maintenance system for information retrieval. In C. T. Yu and C. J. Van Rijsbergen, editors, *SIGIR '87: Proceedings of the 10th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 123 – 131, New York, 1987. ACM Press.

[CO89] Fazli Can and Esen A. Ozkarahan. Dynamic cluster maintenance. *Information Processing and Management*, 25(3):275 – 291, 1989.

[COP03] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In Lawrence L. Larmore and Michel X. Goemans, editors, *STOC '03: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of computing*, pages 30–39, New York, 2003. ACM Press.

[Cow89] Anthony Cowie, editor. *Oxford Advanced Learner's Dictionary*. Oxford University Press, Oxford, 4th edition, 1989.

[CPS98] Yi-Ming Chung, William M. Pottenger, and Bruce R. Schatz. Automatic subject indexing using an associative neural network. In *Proceedings of the 3rd ACM International Conference on Digital Libraries*, pages 59–68, New York, 1998. ACM Press.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1st edition, 1991.

[Dei78] Guntram Deichsel. *Random Walk Clustering in großen Datenbeständen*. PhD thesis, Universität Stuttgart, Stuttgart, 1978.

[DFR+01] Jérôme Darmont, Christophe Fromantin, Stephane Régnier, Le Gruenwald, and Michel Schneider. Dynamic clustering in object-oriented databases: An advocacy for simplicity. In Klaus R. Dittrich, Giovanna Guerrini, Isabella Merlo, Marta Oliva, and Elena Rodríguez, editors, *Objects and Databases, International Symposium, Proceedings*, pages 71–85, Heidelberg, 2001. Springer.

[DG96] Jérôme Darmont and Le Gruenwald. A comparison study of object-oriented database clustering techniques. *Information Sciences*, 94(1–4):55 – 86, 1996.

[DGG48] Allison Davis, Burleigh B. Gardner, and Mary R. Gardner. *Deep South. A Social Anthropological Study of Caste and Class*. University of Chicago Press, Chicago, 1948.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, New York, 2nd edition, 2001.

[Die05] Reinhard Diestel. *Graph Theory*. Springer, Heidelberg, 3rd edition, 2005.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DJ76] Richard Dubes and Anil K. Jain. Clustering techniques: The user's dilemma. *Pattern Recognition*, 8(4):247–260, 1976.

[DO74] Benjamin S. Duran and Patrick L. Odell. *Cluster Analysis – A Survey*. Springer, Heidelberg, 1974.

[DPS98] Jérôme Darmont, Bertrand Petit, and Michel Schneider. OCB: A generic benchmark to evaluate the performances of object-oriented database systems. In Hans-Jörg Schek, Fèlix Saltor, Isidro Ramos, and Gustavo Alonso, editors, *Advances in Database Technology – EDBT'98, 6th International Conference on Extending Database Technology*, pages 326–340, Heidelberg, 1998. Springer.

[Ehr88] Andrew S.C. Ehrenberg. *Repeat-Buying: Facts, Theory and Applications*. Charles Griffin & Company Ltd, London, 2nd edition, 1988.

[EKS+98] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 323–333, San Francisco, 1998. Morgan Kaufmann.

[Eng65] Ora Engelberg. On some problems concerning a restricted random walk. *Journal of Applied Probability*, 2(2):396–404, 1965.

[ER57] Paul Erdös and Alfred Renyi. On random graphs I. *Publicationes Mathematicae*, 6:290–297, 1957.

[ER60] Paul Erdös and Alfred Renyi. On the evolution of random graphs. *Publications of the Hungarian Academy of Sciences*, 5(1):17–61, 1960.

[Fer70] Franz Ferschl. *Markovketten*. Springer, Heidelberg, 1970.

[FGS04] Markus Franke and Andreas Geyer-Schulz. Automated Indexing with Restricted Random Walks on Large Document Sets. In Rachel Heery and Liz Lyon, editors, *Research and Advanced Technology for Digital Libraries – 8th European Conference, ECDL 2004*, pages 232–243, Heidelberg, 2004. Springer.

[FGS05] Markus Franke and Andreas Geyer-Schulz. Using Restricted Random Walks for Library Recommendations. In Gulden Uchyigit, editor, *Web Personalization, Recommender Systems and Intelligent User Interfaces*, pages 107–115, Setúbal, 2005. INSTICC Press.

[FGS07a] Markus Franke and Andreas Geyer-Schulz. A Method for Analyzing the Asymptotic Behavior of the Walk Process in Restricted Random Walk Cluster Algorithm. In *Advances in Data Analysis. Proceedings of the 30th Annual Conference of the German Classification Society (GfKl)*, pages 51–58, Heidelberg, 2007. Springer.

[FGS07b] Markus Franke and Andreas Geyer-Schulz. Using restricted random walks for library recommendations and knowledge space exploration. *International Journal of Pattern Recognition and Artificial Intelligence*, 21(2):355 – 373, 2007.

[FGSN06] Markus Franke, Andreas Geyer-Schulz, and Andreas Neumann. Building recommendations from random walks on library opac usage data. In Sergio Zani, Andrea Cerioli, Marco Riani, and Maurizio Vichi, editors, *Data Analysis, Classification and the Forward Search. Proceedings of the CLADAG 2005*, pages 235 – 246, Heidelberg, 2006. Springer.

[Fis58] Walter D. Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53(284):789–798, 1958.

[Fis87] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[FM82] Charles M. Fiduccia and Robert M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th Conference on Design Automation*, pages 175–181, Piscataway, 1982. IEEE Press.

[FPT81] Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981.

[Fra03] Markus Franke. Clustering of very large document sets using random walks. Master's thesis, Universität Karlsruhe (TH), Karlsruhe, 2003.

[Fre03] Linton C. Freeman. Finding social groups: A meta-analysis of the southern women data. In Kathleen Carley, Philippa Pattison, and Ronald Breiger, editors, *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*, Washington, 2003. National Research Council, National Academies Press.

[FT05] Markus Franke and Anke Thede. Clustering of Large Document Sets with Restricted Random Walks on Usage Histories. In Claus Weihs and Wolfgang Gaul, editors, *Classification – the Ubiquitous Challenge: Proceedings of the 28th Annual Conference of the German Classification Society (GfKl)*, pages 402–409, Heidelberg, 2005. Springer.

[FXZ92] Douglas H. Fisher, Ling Xu, and Nazih Zard. Ordering effects in clustering. In Derek H. Sleeman and Peter Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992)*, pages 162–168, San Francisco, 1992. Morgan Kaufmann.

[GD03] Ruxandra Gorunescu and Dan Dumitrescu. Evolutionary clustering using an incremental technique. *Informatica*, 98(2):25–33, 2003.

[GDN03] Jonathan Gomez, Dipankar Dasgupta, and Olfa Nasraoui. A new gravitational clustering algorithm. In Daniel Barbará and Chandrika Kamath, editors, *Proceedings of the Third SIAM International Conference on Data Mining*, Philadelphia, 2003. SIAM.

[GG04] Chetan Gupta and Robert L. Grossman. GenIc: A single-pass generalized incremental algorithm for clustering. In Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn, editors, *Proceedings of the Fourth SIAM International Conference on Data Mining*, Philadelphia, 2004. SIAM.

[GGH+01] Jie Gao, Leonidas Guibas, John Hershberger, Li Zhang, and An Zhu. Discrete mobile centers. In *SCG '01: Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, pages 188–196, New York, 2001. ACM Press.

[GGH+03] Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Discrete mobile centers. *Discrete and Computational Geometry*, 30(1):45–65, 2003.

[GGSHST02] Wolfgang Gaul, Andreas Geyer-Schulz, Michael Hahsler, and Lars Schmidt-Thieme. eMarketing mittels Recommendersystemen. *Marketing ZFP*, 24:47 – 55, 2002.

[GJ02] Erhard Godehardt and Jerzy Jaworski. Two Models of Random Intersection Graphs for Classification. In Otto Opitz and Manfred Schwaiger, editors, *Exploratory Data Analysis in Empirical*

Research. Proceedings of the 25th Annual Conference of the German Classification Society (GfKl), pages 67–81, Heidelberg, 2002. Springer.

[GLF89] John H. Gennari, Pat Langley, and Douglas H. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40(1–3):11–61, 1989.

[GMM+03] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[GO02] Josephine Griffith and Colm O'Riordan. Non-traditional collaborative filtering techniques. Technical Report NUIG-IT-121002, National University of Ireland, Galway, 2002.

[Goo77] Irving J. Good. The botryology of botryology. In John VanRyzin, editor, *Classification and Clustering: Proceedings of an Advanced Seminar Conducted by the Mathematics Research Center, the University of Wisconsin at Madison*, pages 73–94, New York, 1977. Academic Press.

[GR69] John C. Gower and Gavin J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 18(1):54–64, 1969.

[Gra68] Augustine H. Gray, Jr. Level touchings in a restricted random walk. *SIAM Journal on Applied Mathematics*, 16(6):1123–1129, 1968.

[GSHJ01] Andreas Geyer-Schulz, Michael Hahsler, and Maximillian Jahn. Educational and scientific recommender systems: Designing the information channels of the virtual university. *International Journal of Engineering Education*, 17(2):153 – 163, 2001.

[GSHNT03] Andreas Geyer-Schulz, Michael Hahsler, Andreas Neumann, and Anke Thede. Recommenderdienste für wissenschaftliche Bibliotheken und Bibliotheksverbünde. In Andreas Geyer-Schulz and Alfred Taudes, editors, *Informationswirtschaft: Ein Sektor mit Zukunft*, pages 43 – 57. Gesellschaft für Informatik, Köllen Druck+Verlag GmbH, Bonn, 2003.

[GSNT03] Andreas Geyer-Schulz, Andreas Neumann, and Anke Thede. An architecture for behavior-based library recommender systems – integration and first experiences. *Information Technology and Libraries*, 22(4), 2003.

[GT95] Mario Gerla and Jack Tzu-Chieh Tsai. Multicluster, mobile, multimedia radio network. *Journal of Wireless Networks*, 1(3):255 – 265, 1995.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[GWW99] Yoram Gdalyahu, Daphna Weinshall, and Michael Werman. A randomized algorithm for pairwise clustering. In Michael J. Kearns, Sara A. Solla, and David A. Cohn, editors, *Advances in Neural Information Processing Systems 11, NIPS Conference*, pages 424–430, Cambridge, 1999. The MIT Press.

[Haj88] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of Operation Research*, 13(2):311 – 329, 1988.

[Har75] John A. Hartigan. *Clustering Algorithms*. John Wiley and Sons, New York, 1975.

[HBV02a] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. Cluster validity methods: Part I. *ACM SIGMOD Record*, 31(2):40–45, 2002.

[HBV02b] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. Cluster validity methods: Part II. *ACM SIGMOD Record*, 31(3):19–27, 2002.

[HK89] Scott E. Hudson and Roger King. Cactis: A self-adaptive, concurrent implementation of an object-oriented database management system. *ACM Transactions on Database Systems*, 14(3):291 – 321, 1989.

[HK92] Lars Hagen and Andrew B. Kahng. A new approach to effective circuit clustering. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 422–427, Santa Clara, 1992. ACM and IEEE Computer Society.

[HK01] David Harel and Yehuda Koren. On clustering using random walks. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science*, pages 18–41, Heidelberg, 2001. Springer.

[HK03] Khaled M. Hammouda and Mohamed S. Kamel. Incremental document clustering using cluster similarity histograms. In *2003 IEEE/WIC International Conference on Web Intelligence, (WI 2003)*, pages 597–601, Washington, 2003. IEEE Computer Society.

[HLL94] Kien A. Hua, Sheau-Dong Lang, and Wen K. Lee. A decomposition-based simulated annealing technique for data clustering. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117–128, New York, 1994. ACM Press.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[HP04] Sariel Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31(4):545–565, 2004.

[IIK96] Mary Inaba, Hiroshi Imai, and Naoki Katoh. Experimental results of randomized clustering algorithm. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 1–2, New York, 1996. ACM Press.

[IKI94] Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering (extended abstract). In *SCG '94: Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 332 – 339, New York, 1994. ACM Press.

[Jac08] Paul Jaccard. Nouvelles recherches sur la distribution florale. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 44(163):223–270, 1908.

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264 – 323, 1999.

[Jun99] Dieter Jungnickel. *Graphs, Networks and Algorithms*. Springer Verlag, Heidelberg, 1999.

[Kar] George Karypis. CLUTO – a clustering toolkit. URL: http://glaros.dtc.umn.edu/gkhome/views/cluto. (02.08.2006).

[Kar93] Alan F. Karr. *Probability*. Springer, Heidelberg, 1993.

[Kar03] George Karypis. Cluto: A clustering toolkit. Technical report, University of Minnesota, Minneapolis, 2003.

[KGV83] Scott Kirkpatrick, Charles D. Jr. Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KKG03] Hans-Peter Kriegel, Peer Kröger, and Irina Gotlibovich. Incremental OPTICS: Efficient computation of updates in a hierarchical cluster ordering. In *Proceedings of the 5th International Conference on Data Warehousing and Knowledge Discovery*, pages 244–233, Heidelberg, 2003. Springer.

[KLNS89] Jeff D. Kahn, Nathan Linial, Noam Nisan, and Michael E. Saks. On the cover time of random walks on graphs. *Journal of Theoretical Probability*, 2(1):121–128, 1989.

[KM99] Arnd Kohrs and Bernard Merialdo. Clustering for collaborative filtering applications. In *Computational Intelligence for Modelling, Control & Automation 1999*, pages 199–204, Amsterdam, 1999. IOS Press.

[KR90] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, New York, 1990.

[KS96] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601 – 640, 1996.

[Kun99] Sukhamay Kundu. Gravitational clustering: A new approach based on the spatial distribution of the points. *Pattern Recognition*, 32(7):1149–1160, 1999.

[Kun03] Martin Kunz et al. SWD Sachgruppen. Technical report, Die Deutsche Bibliothek, 2003.

[KVCP97] Prasad Krishna, Nitin H. Vaidya, Mainak Chatterjee, and Dhiraj K. Pradhan. A cluster-based approach for routing in dynamic networks. *SIGCOMM Computer Communication Review*, 27(2):49 – 64, 1997.

[LG97] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[LH03] Boris Lauser and Andreas Hotho. Automatic multi-label subject indexing in a multilingual environment. In Traugott Koch and Ingeborg Torvik Solvberg, editors, *Research and Advanced Technology for Digital Libraries: Proceedings of the ECDL 2003*, pages 140–151, Heidelberg, 2003. Springer.

[LHY04] Yifan Li, Jiawei Han, and Jiong Yang. Clustering moving objects. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 617–622, New York, 2004. ACM Press.

[Lov96] László Lovász. Random walks on graphs: A survey. In Dezsö Miklos, editor, *Combinatorics, Paul Erdös is Eighty*, volume 2, pages 353–397, Budapest, 1996. Janos Bolyai Mathematical Society.

[LR80] John William Strutt Lord Rayleigh. On the resultant of a large number of vibrations of the same pitch and of arbitrary phase. *Philosophical Magazine*, 10:73–80, 1880.

[LR05] John William Strutt Lord Rayleigh. The problem of the random walk. *Nature*, 72(1866):318, 1905.

[LVKG04] Jessica Lin, Michail Vlachos, Eamonn J. Keogh, and Dimitrios Gunopulos. Iterative incremental clustering of time series. In Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari, editors, *Advances in Database Technology – EDBT 2004, 9th International Conference on Extending Database Technology, Proceedings*, pages 106–122, Heidelberg, 2004. Springer.

[LW67a] Godfrey N. Lance and William Thomas Williams. A general theory of classificatory sorting strategies. I. Hierarchical systems. *Computer Journal*, 9(4):373–380, 1967.

[LW67b] Godfrey N. Lance and William Thomas Williams. A general theory of classificatory sorting strategies. II. Clustering systems. *Computer Journal*, 10(3):271–277, 1967.

[Mac67] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, 1967. University of California Press.

[Mah36] Prasanta Chandra Mahalanobis. On the generalised distance in statistics. In *Proceedings of the National Institute of Science of India*, volume 12, pages 49–55, Calcutta, 1936.

[Mar12] Andrei A. Markoff. *Wahrscheinlichkeitsrechnung*. B.G. Teubner, Leipzig, 2nd edition, 1912.

[Mat77] David W. Matula. Graph theoretic techniques for cluster analysis algorithms. In John VanRyzin, editor, *Classification and Clustering: Proceedings of an Advanced Seminar Conducted by the Mathematics Research Center, the University of Wisconsin at Madison*, pages 95–129, New York, 1977. Academic Press.

[MK94] William J. McIver Jr. and Roger King. Self-adaptive, on-line reclustering of complex object data. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 407–418, New York, 1994. ACM Press.

[MMR96] Dario Maio, Davide Maltoni, and Stefano Rizzi. Dynamic clustering of maps in autonomous agents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(11):1080–1091, 1996.

[MR00] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, 2nd edition, 2000.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations for fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[MRSV86] Debasis Mitra, Fabio Romeo, and Alberto Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. *Advances in Applied Probability*, 18(3):747–771, 1986.

[MZ99] A. Bruce McDonald and Taieb Znati. A mobility-based framework for adaptive clustering in wireless adhoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1466–1487, 1999.

[MZ02] A. Bruce McDonald and Taieb F. Znati. Design and simulation of a distributed dynamic clustering algorithm for multimode routing in wireless ad hoc networks. *Simulation*, 78(7):408–422, 2002.

[Nev95] Arthur J. Nevins. A Branch and Bound Incremental Conceptual Clusterer. *Machine Learning*, 18(1):5 – 22, 1995.

[Nic97] David M. Nichols. Implicit rating and filtering. In *Fifth DELOS Workshop: Filtering and Collaborative Filtering*, pages 28–33, Le Chesnay, 1997. ERCIM.

[NUCG03] Olfa Nasraoui, Cesar Cardona Uribe, Carlos Rojas Coronel, and Fabio A. González. TECNO-STREAMS: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, pages 235–242, Washington, 2003. IEEE Computer Society.

[OMM+02] Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings of the 18th International Conference on Data Engineering*, pages 685–694, Piscataway, 2002. IEEE Press.

[Pea05] Karl Pearson. The problem of the random walk. *Nature*, 72(1865):294, 1905.

[PL02] Patrick Pantel and Dekang Lin. Document clustering with committees. In Järvelin and Kalervo, editors, *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 199 – 206, New York, 2002. ACM Press.

[Pol21] Georg Polya. Über eine Aufgabe der Wahrscheinlichkeitsrechnung betreffend die Irrfahrt im Straßennetz. *Mathematische Annalen*, 84:149–160, 1921.

[PS06] Harald Prehn and Gerald Sommer. An adaptive classification algorithm using robust incremental clustering. In *18th International Conference on Pattern Recognition, ICPR 2006*, pages 896–899, Washington, 2006. IEEE Computer Society.

[PZOD99] Srinivasan Parthasarathy, Mohammed Javeed Zaki, Mitsunori Ogihara, and Sandhya Dwarkadas. Incremental and interactive sequence mining. In *Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management*, pages 251–258, New York, 1999. ACM Press.

[REL99] Arnaud Ribert, Abdel Ennaji, and Yves Lecourtier. An incremental hierarchical clustering. In *Conférence Internationale IAPR-VI'99, Vision Interface*, pages 586–591, 1999.

[RKSA00] Lakshmi Ramachandran, Manika Kapoor, Abhinanda Sarkar, and Alok Aggarwal. Clustering algorithms for wireless ad hoc networks. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M 2000)*, pages 54–63, New York, 2000. ACM Press.

[ROS01] Andréa W. Richa, Katia Obraczka, and Arunhaba Sen. Application-oriented self-organizing hierarchical clustering in dynamic networks: A position paper. In *Proceedings of 1st ACM Workshop on Principles of Mobile Computing (POMC)*, pages 57–65, New York, 2001. ACM Press.

[RPH05] Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. Randomized algorithms and NLP: Using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 622–629, Morristown, 2005. Association for Computational Linguistics.

[RT98] Joseph Roure and Luis Talavera. Robust incremental clustering with bad instance orderings: A new strategy. In Helder Coelho, editor, *Progress in Artificial Intelligence – IBERAMIA '98, Sixth Ibero-American Conference on AI*, pages 136–147, Heidelberg, 1998. Springer.

[RV87] James R. Rowland and Gregg T. Vesonder. Incremental conceptual clustering from existing databases. In Pat Davis and Vicki McClin-

tock, editors, *Proceedings of the 15th ACM Annual Conference on Computer Science*, pages 80–87, New York, 1987. ACM Press.

[RV97] Paul Resnick and Hal R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56 – 58, 1997.

[Sal88] Gerald Salton, editor. *Automatic Text Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, 1988.

[Sam38] Paul A. Samuelson. A note on the pure theory of consumer's behaviour. *Economica*, 5(17):61–71, 1938.

[Sam48] Paul A. Samuelson. Consumption theory in terms of revealed preference. *Economica*, 15(60):243–253, 1948.

[Sch02] Joachim Schöll. *Clusteranalyse mit Zufallswegen*. PhD thesis, TU Wien, Wien, 2002.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[SEKX98] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[SFFE01] Giovanni Semeraro, Stefano Ferilli, Nicola Fanizzi, and Floriana Esposito. Document classification and interpretation through the inference of logic-based models. In Panos Constantopoulos and Ingeborg T. Sølvberg, editors, *Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2001*, pages 59–70, Heidelberg, 2001. Springer.

[SH01] Gabriel L. Somlo and Adele E. Howe. Incremental clustering for profile maintenance in information gathering web agents. In Elisabeth André, Sandip Sen, Claude Frasson, and Jörg P. Müller, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 262–269, New York, 2001. ACM Press.

[SJ01] Martin Szummer and Tommi Jaakkola. Partially labeled classification with markov random walks. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14, NIPS 2001*, pages 945–952, Cambridge, 2001. MIT Press.

[SKKR02] Badrul M. Sarwar, George Karypis, Joseph Konstan, and John Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the Fifth International Conference on Computer and Information Technology*, Bangladesh, 2002.

[SKR01] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, 2001.

[Sne57] Peter H.A. Sneath. The application of computers to taxonomy. *The journal of general microbiology*, 17:201–226, 1957.

[Sok77] Robert R. Sokal. Clustering and classification: Background and current directions. In John VanRyzin, editor, *Classification and Clustering: Proceedings of an Advanced Seminar Conducted by the Mathematics Research Center, the University of Wisconsin at Madison*, pages 1–15, New York, 1977. Academic Press.

[SP02] Joachim Schöll and Elisabeth Paschinger. Cluster Analysis with Restricted Random Walks. In Krzysztof Jajuga, Andrzej Sokolowski, and Hans-Hermann Bock, editors, *Classification, Clustering, and Data Analysis. Proceedings of the 8th Conference of the International Federation of Classification Societies (IFCS-2002)*, pages 113–120, Heidelberg, 2002. Springer.

[Spe01] Joel H. Spencer. *The Strange Logic of Random Graphs*. Springer, Heidelberg, 2001.

[Spi01] Frank Spitzer. *Principles of Random Walk*. Springer, Heidelberg, 2nd edition, 2001.

[SS73] Peter H. A. Sneath and Robert R. Sokal. *Numerical Taxonomy: The Principles and Practice of Numerical Classification*. Freeman, San Francisco, 1973.

[SS05] Yogish Sabharwal and Sandeep Sen. A linear time algorithm for approximate 2-means clustering. *Computational Geometry: Theory and Applications*, 32(2):159 – 172, 2005.

[SSP03] Joachim Schöll and Elisabeth Schöll-Paschinger. Classification by restricted random walks. *Pattern Recognition*, 36(6):1279–1290, 2003.

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[TS01] Naftali Tishby and Noam Slonim. Data clustering by markovian relaxation and the information bottleneck method. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13, NIPS 2000*, pages 640–646, Cambridge, 2001. MIT Press.

[vD00a] Stijn van Dongen. A Cluster Algorithm for Graphs. Technical report, Centrum voor Wiskunde en Informatica, Amsterdam, 2000.

[vD00b] Stijn van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, Utrecht, 2000.

[vD00c] Stijn van Dongen. Performance criteria for graph clustering and markov cluster experiments. Technical report, National Research Institute for Mathematics and Computer Science in the Netherlands, Amsterdam, 2000.

[Vie97] Johannes Viegener. *Inkrementelle, domänenunabhängige Thesauruserstellung in dokumentbasierten Informationssystemen durch Kombination von Konstruktionsverfahren*. infix, Sankt Augustin, 1997.

[Wan97] Ke Wang. Discovering patterns from large and dynamic sequential data. *Journal of Intelligent Information Systems*, 9(1):33 – 56, 1997.

[War63] Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[Wat81] Satosi Watanabe. Pattern recognition as a quest for minimum entropy. *Pattern Recognition*, 13(5):381–387, 1981.

[WF00] Wai-chiu Wong and Ada Wai-chee Fu. Incremental document clustering for web page classification. In *Proceedings of the International Conference on Information Society in the 21st Century*, Piscataway, 2000. IEEE Press.

[WIY02] Dwi H. Widyantoro, Thomas R. Ioerger, and John Yen. An incremental approach to building a cluster hierarchy. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 705–708, Piscataway, 2002. IEEE Press.

[WK99] Fa-Yueh Wu and Hervé Kunz. Restricted random walks on a graph. *Annals of Combinatorics*, 3:475–481, 1999.

[WL93] Zhenyu Wu and Richard M. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101 – 1113, 1993.

[Wri77] William E. Wright. Gravitational clustering. *Pattern Recognition*, 9:151–166, 1977.

[WWP88] Shijie J. Wan, S. K. Michael Wong, and Przemyslaw Prusinkiewicz. An algorithm for multidimensional data clustering. *ACM Transaction on Mathematical Software*, 14(2):153 – 162, 1988.

[XYZ06] Yunpeng Xu, Xing Yi, and Changshui Zhang. A random walks method for text classification. In *Proceedings of the 2006 SIAM Conference on Data Mining*, pages 338–345, Philadelphia, 2006. SIAM.

[Yan99] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1):69–90, 1999.

[Yan05] Shi-Jie Yang. Exploring complex networks by walking on them. *Physical Review E*, 71(016107):1–5, 2005.

[YM98] Clement T. Yu and Weiyi Meng. *Principles of Database Query Processing for Advanced Applications*. Morgan Kaufmann, San Francisco, 1998.

[Zho05] Shi Zhong. Efficient streaming text clustering. *Neural Networks*, 18(5–6):790 – 798, 2005.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 103–114, New York, 1996. ACM Press.

Cluster algorithms are a valuable tool in applications where large data sets (more than one million objects) must be grouped for a closer analysis, for instance in marketing or for information services like recommender systems. In an increasingly dynamic environment, access to up-to-date clusterings with a high quality becomes more and more important. Unfortunately, this is, at least for large data sets, not possible with standard cluster techniques.

The main contribution of the thesis is an extension of the restricted random walk (RRW) clustering algorithm by Schöll and Paschinger that allows real time updates of the cluster structure while guaranteeing the same cluster quality as the static version. RRW clustering is a randomized algorithm based on the execution of a series of finite random walks on an object set with a similarity relation between its members. The walks are subject to the restriction that each node must be more similar to its successor than to its predecessor in the walk, such that the similarity on subsequent edges of the walk increases monotonically until no more nodes can be found that satisfy the restriction.
The principle for the cluster construction is that a pair of objects that has a high position in one of the walks also has a strong tendency to belong to the same cluster. Using this assumption, clusters are generated in real time from the accumulated walk data.

If two objects change their similarity during an update, this may cause existing walks to become invalid, as the restriction on the similarity of subsequent objects no longer holds, or new possibilities for walks may arise. Both cases can be handled efficiently by appropriately truncating existing walks and restarting them from the place where they were truncated.
The thesis proves that the algorithm maintains a walk data set, and thus a clustering, whose members have the same occurrence probabilities as the members of a walk data set created by the execution of the static algorithm on the modified similarities. Thus, there is no need, contrarily to many other dynamic cluster algorithms, for a complete reclustering of the data set to maintain the quality of the clusters.

ISBN: 978-3-86644-183-5

www.uvka.de