DISS. ETH NO. 19198

Understanding and Organizing User Generated Data: Methods and Applications

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

MICHAEL KUHN

MSc ETH ITET, ETH Zürich

born 12.02.1979

citizen of
Waltenschwil (AG)

accepted on the recommendation of

Prof. Roger Wattenhofer, examiner
Prof. Albrecht Schmidt, co-examiner

2010

Abstract

The Internet in general, and the Web 2.0, together with the trend towards mobile terminals in particular, have recently had an immense impact on the society. As a result of these developments, users nowadays produce data on an unprecedented scale. This new kind of data has attracted the attention of researchers from a wide variety of disciplines. In this thesis we contribute some small pieces to the huge interdisciplinary efforts to understand this implicitly and explicitly generated data. More importantly, we put these analytic efforts into a practical context by making the insights directly accessible to the end-users by means of concrete applications.

In the first part of the thesis we investigate the interconnections between people in (online) social networks. Based on some relevant properties of these networks, we then propose two mobile applications, one to assist a user who wants to address a group of people from within a resource restricted device, and one that unobtrusively searches for potential friends while the user is pursuing everyday activities.

The second part of the thesis studies the extraction of similarity measures from user generated content. We thereby focus on two main domains: Scientific conferences and music. We show that a large collection of publication records implicitly contains information about different aspects of conference similarities. Two of these aspects – quality and thematic scope – form the fundamental building blocks of *confsearch*, a conference search engine we have implemented. Our conference similarity measure is further used to draw a map of conferences using a graph embedding algorithm and to discuss the world of conferences in a playful manner.

Towards the end of the thesis we discuss different facets of music similarity and its use in end-user applications. In particular, we take advantage of the fact that the cumulated listening histories of a large user basis contain valuable information about the similarity of songs. The resulting similarity measure is known to better reflect the users' perception than state-of-the-art audio based methods. However, it lacks a compact representation as known from audio based techniques, a fact that greatly complicates the design of intuitive user interfaces. To overcome this problem, we pick up the idea of a map, as already discussed in the context of scientific conferences. In particular, we propose to compactly embed our usage data based music similarity measures into a Euclidean space. For this purpose, we make use of two different techniques, one that solely relies on the listening behavior, and one that combines this data with the more explicit information contained in social tags. We finally demonstrate the practical usefulness of the concept of a music map in a comprehensive mobile music player for the Android platform that gained a remarkable popularity. In a user study, we show that the integrated similarity aware features are frequently used. From this study, and from user comments, we conclude that our map in fact facilitates the design of similarity based music retrieval interfaces, and that such interfaces are also well accepted by the community.

Zusammenfassung

Das Internet im allgemeinen, und im speziellen das Web 2.0 zusammen mit dem Trend Richtung mobiler Endgeräte haben in den letzten Jahren unsere Gesellschaft entscheidend geprägt. Also Folge dieser Entwicklungen werden heute Nutzerdaten in nie dagewesener Grössenordnung generiert und gesammelt. Diese neue Art von Daten hat die Aufmerksamkeit einer Vielzahl von Wissenschaftlern aus den verschiedensten Fachrichtungen auf sich gezogen. Diese Arbeit leistet einen kleinen Beitrag zu den umfangreichen interdisziplinären Bemühungen, diese implizit und explizit generierten Daten zu verstehen. Unsere analytischen Anstrengungen sind nicht rein theoretischer Natur sondern haben auch einen praktischen Hintergrund. Insbesondere setzen wir jeweils die gewonnenen Erkenntnisse in konkrete Anwendungen um, welche dem Endbenutzer direkt zur Verfügung stehen.

Im ersten Teil der Arbeit untersuchen wir die Verbindungen zwischen Personen in (virtuellen) sozialen Netzwerken. Aufbauend auf verschiedenen Eigenschaften dieser Netzwerke schlagen wir dann zwei mobile Applikationen vor. Die erste Applikation hilft dem Benutzer, eine ganze Gruppe von Kontakten von einem kleinen Endgerät aus zu selektieren. Die andere Anwendung sucht im Hintergrund nach potentiellen Freunden für seinen Besitzer, während dieser seinen Alltagsbeschäftigungen nachgeht.

Der zweite Teil der Arbeit beschäftigt sich damit, verschiedene Ähnlichkeitsmasse aus benutzergenerierten Daten zu extrahieren. Dabei konzentrieren wir uns auf zwei Gebiete: Wissenschaftliche Konferenzen und Musik. Wir zeigen, dass die Meta-Daten einer Vielzahl von Publikationen implizite Informationen über verschiedene Aspekte von Konferenzähnlichkeiten besitzen. Aufbauend auf diesen verschiedenen Ähnlichkeitsaspekten – insbesondere bezüglich Qualität und thematischer Ausrichtung einer Konferenz – haben wir die Konferenzsuchmaschine *confsearch* implementiert. Des weitern haben wir unser Ähnlichkeitsmass verwendet, um mittels eines Graphen-Einbettungs-Algorithmus' eine Karte von wissenschaftlichen Konferenzen zu zeichnen, und um in spielerischer Weise über die Informatik-Welt zu sinnieren.

Gegen Ende der Arbeit gehen wir über zu Musikähnlichkeit und diskutieren insbesondere den Gebrauch solcher Information in Applikationen. Dabei nutzen wir aus, dass die kummulierten Aufzeichungen des Hörverhaltens vieler Anwender wertvolle Informationen über die Ähnlichkeit von Liedern beinhalten. Es ist bekannt, dass Ähnlichkeitsmasse, welche auf sochen Nutzerdaten basieren, die Empfindung der Anwender besser wiederspiegeln als die aktuell besten audiobasierten Methoden. Allerdings verfügen solche subjektiven Masse nicht über eine gleichermassen Kompakte Darstellung, wie dies für audiobasierte Ansätze typischerweise der Fall ist – eine Tatsache welche das Entwickeln von intuitiven Benutzerschnittstellen ungleich schwieriger macht. Um diesem Problem Herr zu werden, greifen wir die Idee einer Karte, wie schon im Bereich von Konferenzen behandelt, wieder auf. Insbesondere schlagen wir vor, unser nutzerdatenbasiertes Ähnlichkeitsmass in kompakter Form in einen Euklidischen Raum einzubetten. Zu diesem Zweck verwenden wir zwei unterschiedliche Techniken, eine welche ausschliesslich das Hörverhalten des Nutzer einbezieht, und eine zweite, welche diese Daten mit den expliziteren Informationen von sozialen Tags kombiniert. Schliesslich demonstrieren wir die praktische Anwendbarkeit des Konzeptes einer Musikkarte in einem umfangreichen mobilen Musikplayer für die Android Plattform, welche sich bereits einer beachtlichen Popularität erfreut. Eine Benutzer-Studie zeigt, dass die eingebauten ähnlichkeitsbasierten Features rege benutzt werden. Aus diesen Beobachtungen schliessen wir, dass unsere Musikkarte das Entwerfen von ähnlichkeitsbasierten Benutzerschnittstellen tatsächlich vereinfacht, und dass die resultierenden Oberflächen von den Benutzern beachtet und akzeptiert werden.

Acknowledgements

This thesis would not have been possible without the great support from a variety of people. First of all, I would like to thank my advisor Roger Wattenhofer for guiding me through my thesis with patience, smart ideas, and lots of valuable discussions. It was a great pleasure to get the opportunity to explore a field new to the research group from a practical perspective.

I would also like to thank my co-examiner Albrecht Schmidt for investing the time to read through the thesis and for serving my committee board.

Furthermore, I want to emphasize the warm and inspiring environment provided by the research group. For this great work environment, I want to express my gratitude to all the members of Da Cool Gang and to all Da Incredibly Smart Computer Officers. I especially want to thank Olga Goussevskaia and Samuel Welten for being great office mates who did not mess with my mess, helped ordering my music, and cha(lla)nged my Russian skills in a positive way. Moreover, I want to thank my brother Fabian for connecting me with the group, Aaron Zollinger for making Google a bit less evil, Nicolas Burri for playing more table soccer games than me, Thomas Moscibroda for telling me about “the angle”, Keno Albrecht for showing me what an authentic Keno looks like, Regina O’Dell for increasing the population, Pascal von Rickenbach for always having concise advice, Stefan Schmid for supplying the group with a trophy, Roland Flury for making Google even less evil, Thomas Locher for making Christoph discussing clock-sync with him rather than the group, Raphael Eidenbenz for being the best Heizkörper in our corridor, Remo Meier for never saying anything bad, Yvonne-Anne Oswald for representing Museek with her voice and dancing, Benjamin Sigg for being the first vegetarian to invite me to a BBQ, Johannes Schneider for co-representing our group in the ice-rink, Christoph Lenzen for winning some of his (many) ELO-points with me, Philipp Sommer for sensibly sensing our environment, Jasmin Smula for introducing me to HIMYM, Tobias Langner for jointly enjoying beer and football – albeit being a fan of (the wrong) FCB, Stephan Holzer for sharing my affinity to meat, Barbara Keller for teaching me how to motivate students in a sweet way, and Yuval Emek for spending some of his ELO-points to me.

Moreover, it was a great pleasure to work together with all the guys who were looking after the systems that are hosting my projects. Thanks to Andreas Wetzel, Thierry Dussuet, Roland Mathias, Damian Friedly, and Thomas Steingruber for bringing our servers up faster than me bringing them down.

Last but not least I am really grateful to my family for supporting me in so many ways. Thanks to my brother Fabian for being a great brother who influenced my educational history from the early days. And, of course, many thanks to my parents Anne-Marie and Herbert for all the love and support I could enjoy throughout my entire life. Finally, I want to express my deepest feelings to my darling Tanja for accepting me as I am and for always encouraging me in what I am doing.

Contents

Chapter 1

Introduction

In the very beginning of the human existence, the main function of social groups presumably was the increased chance of survival. Over time, the driving forces for interactions in social groups, and thus also group-structures, have changed. With the beginning of writing, the invention of the wheel, and the domestication of the horse, the bonds of locality started to disrupt and allowed longer distance relationships to appear. In the 15th century, Gutenberg laid the foundation for mass media with the invention of printing. The creation, transportation and processing of information entered new dimensions in terms of quantity and time. Together with trains, cars and airplanes, information technology in the shape of newspapers, telegraphs, radio, television and the Internet transformed huge geographical distances into tiny fragments of time. In 1964, Marshall McLuhan coined the metaphor "Global Village" and wrote [79]:

"As electrically contracted, the globe is no more than a village"

He argues that due to the almost instantaneous reaction times of new ("electric") technologies, each individual inevitably feels the consequences of his/her actions and thus automatically deeply participates in the global society.

McLuhan by then understood, what we now can directly observe – real and virtual world are moving together. He realized that the transmission medium, rather than the transmitted information is at the core of change, as expressed by his famous phrase "the medium is the message". Today the most prevalent communication medium is the Internet. Its effects on society are ubiquitous and clearer than ever confirm McLuhan's words.

By means of e-mail, instant messengers, and social platforms, such as Facebook, MySpace, or Twitter, the Internet has crucially altered the way people interact. In the beginning of the web era, content was static, and the user was a pure consumer. Then, people increasingly started to get part of the game and got involved not only as consumers, but also as producers of content – the Web 2.0 was born. The rapidly growing availability of powerful mobile devices further adds to this trend. In 2008, more users accessed the web from mobile terminals than from traditional desktop computers.[1] Moreover, it is said that everybody earning more than 5 USD a day will eventually possess his/her own mobile phone and thus populate the square that connects cyberstreets with concrete roads. As a researcher and application developer, it is of utmost importance to consider these developments to make sure we do not miss the needs of the next generation.

As a result of these developments, people nowadays generate data on an unprecedented scale. The nature of this data differs from the well structured information we once used to store in databases, and poses many challenges to the research community. There are various research efforts that try to analyze, organize, and understand this new kind of data. People from several fields, such as knowledge discovery, data mining, information retrieval, and recommendation systems intensively work on novel techniques that match the requirements of Web 2.0 data. This thesis on the one hand adds some small pieces to these gigantic data analysis efforts, and on the other hand makes use of these pieces and a variety of important third party results to propose new user interfaces to browse and access different kinds of web data. To account for the trend towards mobile devices, many of these interfaces are proposed for, and integrated into mobile applications.

In the first part of the thesis, we discuss different aspects related to complex graph structures with a particular focus on social networks and friendship links. We start by comparing some recent models that describe the small world phenomenon often observed in such network structures to real world data, namely the Wikipedia[2] article graph and the LiveJournal[3] friendship network. We then propose two applications that take advantage of some of the properties of social networks, one to foster group interaction, and the other for serendipitous friend finding. Both applications are designed for mobile platforms and thus well account for the trend towards a mobile web.

The second part of the thesis focuses on extracting similarity measures from user generated data. Our work in this area concentrates on two fields: The relationship among scientific conferences and the relationships among songs. We propose measures to denote similarity in the world of conferences and in the music universe. Based on these measures, we introduce a conference search engine and a map of music which can serve as the basis for

[1]Source: http://en.wikipedia.org/wiki/Mobile_Web
[2]http://www.wikipedia.org
[3]http://www.livejournal.com

various music retrieval interfaces. To demonstrate the advantages of such a map we have implemented a comprehensive mobile music application which is presented in the end of the thesis.

Part I

Contacts and Ties

Chapter 2

Introduction

In 1967, Stanley Milgram, a well known American Psychologist, conducted an experiment: He asked People from Omaha and Wichita to forward a letter to a target person in Boston. However, people were not allowed to directly send the letter to the target person. Rather, they had to pass it to somebody they knew on first name basis and that they thought to have a higher probability to know the target person. This process was repeated, until somebody knew the target person, and could deliver the letter. The number of hops it took for a letter to arrive was astonishingly small. The observation that the entire population is connected by short acquaintance chains got later popularized by the terms “six degrees of separation” and “small world”.

In an attempt to explain the discovered small world phenomenon, a variety of models have been proposed. Early models were mainly based on random graphs. While they were able to describe the short path lengths, they failed to explain other properties. In particular, such short paths do not only exist, but can also be found by people that have only local knowledge about the network. Moreover, social networks have shown to exhibit high local connectivity, that is, people who share a common friend are likely to be friends themselves. This local clustering property is commonly quantified by the *clustering coefficient*, which measures the probability that nodes that share a neighbor are neighbors themselves. Watts and Strogatz [125] proposed a graph model that was able to describe both, the local clustering as well as the short distances. However, their model still failed to explain the navigability phenomenon, i.e. how short paths can be discovered when having only local knowledge about the graph topology. This gap was later closed by the augmented grid model of Kleinberg [56]. The model takes the geographic properties of the real world into account and proposes a specific edge length distribution for random edges inserted into a grid.

Many of the observed properties, such as short paths or high clustering, appear in a wide variety of networks not restricted to the social context. Typically, such networks grow, in some sense, naturally, i.e. they are not engineered, but emerge in a non, or only loosely controlled way. Examples are the web graph, road and airline networks, neural networks, and protein interaction networks. Such networks also often tend to build hubs, resulting in a power-law degree distribution. This phenomenon has been addressed by the preferential attachment model of Barabasi and Albert [10] that exhibits the observed degree distribution as well as short path lengths. However, it does neither address the navigability phenomenon nor adequately explain the clustering characteristics.

Later models that build upon the pioneering ideas of Watts and Strogatz [125] and Kleinberg [56] incorporate the concepts of hierarchies and social dimensions [124, 57]. While hierarchies extend the geographic foundation of Kleinberg's grid model, the idea of social dimensions accounts for findings concerning the reasons why a person was chosen as the next hop in Milgram's experiment. In the next chapter, we will focus on such models and provide empirical evidence for the existence of both, hierarchic structures as well as social dimensions. Our analysis is not restricted to social networks, but, in case of hierarchies, also includes the Wikipedia article graph, which suggests that these concepts can be observed in a variety of contexts.

The pure understanding of complex networks is surely fascinating. However, the driving force behind research is to benefit from the gained insights in one or the other way. Such a benefit is clearly achieved when end users can directly profit from the results in form of applications. The knowledge about navigability has been used in different contexts. Examples are applications in peer-to-peer systems [129, 102] or in focused web crawling, such as discussed in [80]. In the following we will present two applications that, similarly as these examples, take advantage of insights gained from research on social networks. As opposed to the discussed examples, we take advantage of the local clustering, rather than navigability. Our systems are designed to directly foster social interaction in the mobile domain and thus well account for the trend towards mobile social networking.

The first application, Cluestr, exploits the social dimensions in conjunction with the high local clustering to provide efficient means for group interaction on mobile devices. Cluestr exhibits a contact recommendation interface that significantly speeds up group initialization and thus renders the tedious manual grouping of contacts obsolete. The high clustering coefficients typically observed in social networks indicate that a person's friend of a friend is likely to become this person's friend, too. The second application, *VENETA*, makes use of this observation: It facilitates serendipitous friend-of-friend detection in a decentralized and privacy preserving manner.

Chapter 3

How Things are Connected

The small world effect is best known from the social context. However, it has been shown to occur in other contexts as well, typically in complex networks that are in some sense growing naturally. Thus, the study of the small world phenomenon has received a great deal of interest in different communities. Social scientists analyze complex structures of social networks arising in the context of organizations. Biologists study the interactions of cells in intricate metabolic processes. Computer scientists face the emergence of gigantic online systems, ranging from online social networks, such as LiveJournal, to online collaborative data repositories, such as Wikipedia, and, of course, the underlying technologies, such as the Internet and the world-wide-web.

After Milgram's famous "six degrees of separation" experiment, many models have been proposed in an attempt to capture the discovered small world property. The first models were mainly based on random graphs. Soon, however, people understood that random graphs can only explain the short path lengths, but fail to reflect the high local clustering, and the fact that short paths can be found even if only local knowledge of the graph is available for routing. The work of Kleinberg [56] could fill this gap from a theoretical point of view. A study of Liben et al. [69] on *LiveJournal* data basically confirms the theoretical findings. However, the study shows that, rather than considering pure geographic distances, the density of nodes has to be taken into account, too. Another interesting result of Liben et al. is that approximately a third of the links is independent of geography. Comparable numbers are reported in a similar study [60] that states that 70% of friendships can be "explained" by geographic location and interests of the users. Moreover, Dodds et al. [23] found that geography and occupation were the most important reasons to select the next hop in an e-mail based Milgram like experiment. Other reasons include similar educational background, the sheer number of friends, and the travelling habits of a person.

More recent models are trying to account for the reported non-geographic edges. In particular, such models introduce the notion of hierarchies and social dimensions. The main idea is that individuals cluster the world hierarchically into categories. The deeper a category is in such a hierarchy, the more specific it is (in a profession hierarchy, for example, *Java Programmer* is more specific than *Computer Scientist*). The models further state that the social world can be clustered in more than one way (e.g. by geography, by occupation, or by interests). Watts et al. [124] refer to these different classes of categorization as *social dimensions.* They assume that each social dimension can be represented by an independent hierarchy. As a result, a node's identity can be defined as a multi-dimensional coordinate vector, in which each coordinate represents its position in a certain social dimension. Watts et al. then define a measure of similarity (the *social distance*) as the minimum ultra-metric distance over all dimensions between two nodes. The intuition is that closeness in only one dimension is enough to connote affiliation. An interesting consequence of this measure is that social distance violates the *triangular inequality.* That is, persons A and B might be considered socially close because they are both working in the same specific field, and persons B and C might be considered socially close because the are living in the same building. However, this does not imply closeness between persons A and C.

In this chapter we provide evidence for the presence of hidden hierarchic structures in complex networks, even in a non-social setting. Further, we generalize the notion of social dimensions and introduce the concept of *layers.* A *layer* basically is a subgraph that contains all the edges generated due to one particular reason. We will provide evidence for such layers and see that it is useful to understand the reasons behind object relations prior to applying techniques like classification, or clustering. The more diffuse these reasons are in a particular data set, the more difficult it becomes to develop techniques to automatically analyze the structure of the data. We therefore propose to separate the different layers of a graph and provide an approach to achieve such a separation if some information about the layered structure is known.

The two model features (layers and hierarchies) are validated using two different real world networks: the *Simple English Wikipedia* and the *LiveJournal* online social network. For *Wikipedia*, we show that the average path length between articles is related to the height of their least common ancestor in the category tree, which is taxonomy of the topics in the encyclopedia. Moreover, we show that the category tree can be used to greedily find short paths in the graph using only local information. That is, for efficient routing only the underlying taxonomy, the destination node, the current location and the direct neighbors at this location need to be known. These findings confirm the relationships between hierarchies and graph structure as conjec-

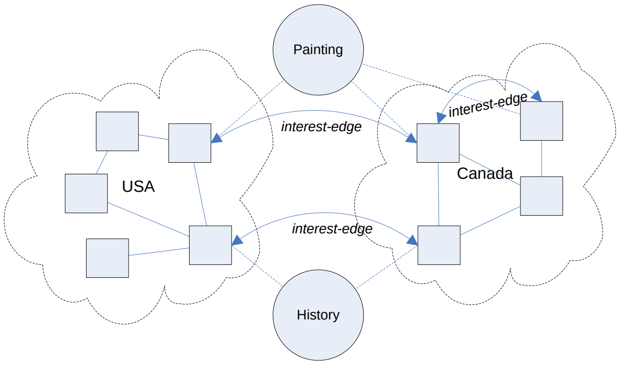


Figure 3.1: *LiveJournal*: Edges belong to different layers (geography, interests).

tured by the models. For the *LiveJournal* social network, we show that the average geographical distance between friends is related to their number of common interests.[1] Surprisingly, the higher the number of common interests, the higher is the average geographical distance between friends. This observation can be seen as empirical evidence for the existence of social dimensions, or layers, in the graph. We take advantage of these findings to improve the precision of the estimation of the location of nodes that lack coordinate information. Moreover, in Chapter 8 we will see an example in which the idea of layers is used in the context of automated conference rating.

3.1 Model

Throughout this chapter we will assume that a naturally grown network exhibits two features:

- *Layers:* A layer is a subgraph containing all the edges that can be explained by a certain reason (see Figure 3.1).
- *Hierarchies:* Each node in a graph belongs to one or more categories. These categories are hierarchically organized and can be represented by a tree.

[1] Members of *LiveJournal* are linked to other members through "friendship" links, and also can declare in their profiles to participate in areas of interest (for details see Section 3.3).

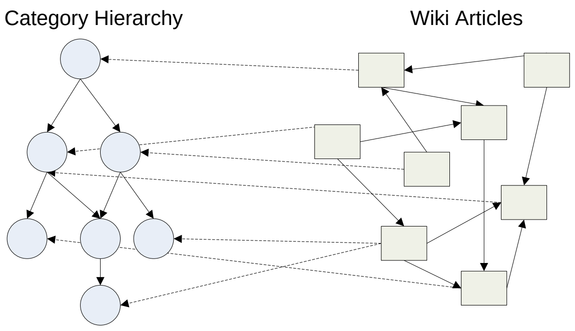


Figure 3.2: *Wikipedia*: The articles are classified according to categories in the category DAG.

Note that these concepts are not independent. Rather, the semantic layers can often be hierarchically classified into groups of increasing granularity. Two persons, both interested in glass painting, for example, are more likely to get acquainted due to their hobby (or profession) than a person interested in glass and another person interested in fresco painting.

3.2 Wikipedia

In this section we show that the *Simple English Wikipedia*[2], which we refer to just as *Wikipedia*, can be modeled using the aforementioned hierarchical approach. We then take advantage of this representation, showing that greedy routing can be applied to efficiently find short paths in the graph.

The *article graph* of *Wikipedia* consists of articles (nodes) and the hyperlinks between articles (edges). In addition to the direct linkage of articles, *Wikipedia* contains a structured organization of topics, the *category tree*. As mentioned before, the category tree is a hierarchical representation of topics that subdivides coarse grained general terms (such as *History*) into ever finer grained terms (such as *History of America* or *History of Oceania*) and finally reaches very specialized categories (such as *Physicians in the American Revolution*, or *Political leaders of the American Revolution*). In fact, the category tree is rather a directed acyclic graph (DAG) than a tree in the case of *Wikipedia*, as nodes can contain more than one parent. Figure 3.2

[2]`http://simple.wikipedia.org`

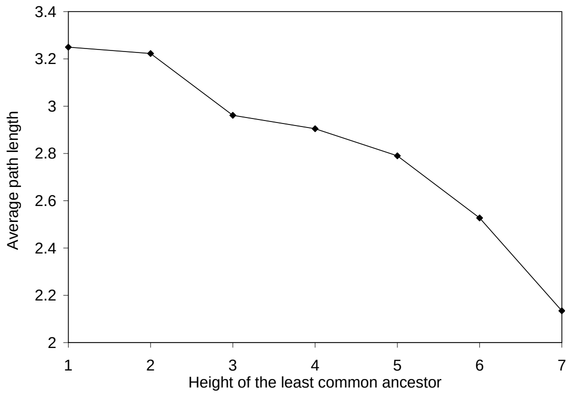


Figure 3.3: The deeper in the category DAG is the LCA, the shorter is the average path length between two articles.

schematically illustrates the relationship between such a category DAG and the Wikipedia articles.

We define the similarity $sim(C_i, C_j)$ between two categories C_i and C_j to be the height of their *Least Common Ancestor* (LCA) in the category DAG (we thereby assume that the root of the DAG has height zero):

$$sim(C_i, C_j) = height(LCA_{DAG}(C_i, C_j)). \tag{3.1}$$

Further, we define $C(A_k)$ to be the set of all categories which an article A_k belongs to. Thereafter, we define the similarity $sim(A_k, A_l)$ between two articles to be the maximum similarity among all pairs of categories to which articles A_k and A_l belong:

$$sim(A_k, A_l) = \max_{C_i \in C(A_k), C_j \in C(A_l)} (sim(C_i, C_j)) \tag{3.2}$$

3.2.1 Evidence for the Hierarchy-Model

The category graph of *Wikipedia* closely resembles the hierarchical structures Kleinberg [57] and Watts et al. [124] introduced in their small-world models. We thus decided to verify whether the models well describe the relations between the *Wikipedia article graph* and the *category DAG*, both naturally grown structures taken from the real world.

If the models are indeed correct, then nodes that are "closer" in the tree should exhibit a higher connectivity, and consequently the path length be-

```
Require: Category DAG, destination article $A_d$, set of articles $P$ visited
    until this point;
 1: $d_{min} = \max_{C_i \in A_d}(height(C_i))$;
 2: Choose neighbor $A_k$ with maximum $sim(A_k, A_d)$;
 3: if $(A_k = A_d)$ then
 4:     Halt;
 5: end if
 6: if $(A_k \in P)$ then
 7:     $A_k$ = random neighbor; (to avoid cycles)
 8: end if
 9: if $(sim(A_k, A_d) < d_{min})$ then
10:     Forward message to $A_k$;
11: else
12:     Start Flooding; $(sim(A_k, A_d) = d_{min}) \Rightarrow$ arrived at the lowest category
        subtree of $A_d$ and greedy cannot make the distance shorter anymore.
13: end if
```

Algorithm 3.1: Greedy Algorithm for *Wikipedia*

tween such nodes should be shorter. We therefore constructed an experiment that compares the average path length between articles to the height of their least common ancestor (LCA) in the category DAG. The average path length for each LCA-value was calculated upon a random sample of 20K pairs of articles. Figure 3.3 summarizes the results. We can observe that the higher the similarity between two articles, i.e., the deeper in the category DAG their LCA, the shorter the average path between them becomes. This confirms the expectations and shows that the model well reflects the real world structure.

3.2.2 Greedy Routing

We take advantage of the fact that the category tree influences path lengths between articles, as demonstrated in the previous section, to implement greedy routing in *Wikipedia*. Figure 3.3 suggests that the distance in the category DAG can be used for greedy routing, since short distances in the category graph indicate short distances in the article link structure. Our experiment consists in implementing a simple routing algorithm (see Algorithm 3.1) that uses the category graph to find short paths in the article graph. The only data that a node requires to forward a message is the similarity measure $sim(A_k, A_d)$, defined in Equation (3.2), between each of its neighbors A_k and the destination article A_d. The greedy step consists in forwarding the message to the neighbor with highest similarity to the destination. This process continues until an article in the closest category to the

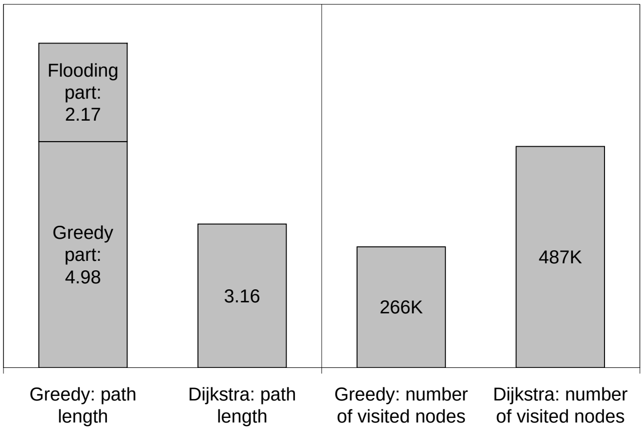


Figure 3.4: Greedy routing in Wikipedia.

destination is reached. Since the greedy step cannot go beyond the granularity of the category graph, once the closest category is reached, a normal flooding is applied to find the destination article among all articles that belong to that category.

We compare the performance of the greedy algorithm to a Dijkstra-based alternative. Note that the algorithm of Dijkstra for finding shortest paths is basically the same as flooding in the case of unweighted graphs. The evaluation of the greedy routing implementation focuses on the following three attributes:

- *Path stretch*: The factor by which the greedy-path gets longer than the shortest path.
- *Flooding stretch*: The factor by which the flooding part of the path becomes shorter.
- *Number of nodes visited*: The relation of the number of nodes visited using flooding and the number of nodes visited using greedy routing (including the final flooding part). Nodes that are visited multiple times are thereby counted multiple times.

In Figure 3.4 it is shown that the average path length obtained by the greedy routing was 7.15 hops, whereas the average shortest path was 3.16. This low *path stretch* indicates that it is indeed possible to find short paths with such an approach. A further analysis of the results in Figure 3.4 shows that, on average, greedy forwarding performed 4.98 steps, and the remaining 2.17 hops result from flooding. Compared to the length of the average shortest path, this gives a *flooding stretch* of 0.69, which results in 45% less nodes that need to be explored in order to reach the destination.

These values show that greedy routing can be an interesting option in graphs for which a hierarchical structure such as the *Wikipedia's* category DAG is known. The average path length increases only by a small constant. The overhead of visiting a huge number of nodes in the graph using flooding algorithms can, on the other hand, be drastically reduced.

For greedy routing to become applicable, it is important to be able to quickly calculate the (category DAG-) distance between two nodes. For our experiments we decided to precompute all the pairwise distances (i.e. LCAs) between categories. However, if the number of categories becomes large this approach becomes infeasible, since the corresponding table grows with $O(n^2)$. For larger category hierarchies, graph labeling provides an elegant alternative. An intelligent label l_i is attached to each category c_i. The labels are chosen such that they are short and that a distance function $d(l_i, l_j)$ can quickly be evaluated.

3.3 Live Journal

In this section we show that an online social network, such as *LiveJournal*, can be modeled using the layered approach described in Section 3.1. Furthermore, we show that if one of the layers is filtered out from the data as a preprocessing step, a better location estimation can be obtained on the remaining graph.

LiveJournal is a popular social networking site that currently counts about 18 million users.[3] As usual for social networking services, these users are connected to each other by friendship relations and thereby form a social graph. Further, *LiveJournal* users can create and participate in interest groups, and they can indicate their place of residence. As opposed to many similar services, *LiveJournal* is freely crawlable, which allows to retrieve and study the aforementioned friendship relations and interest memberships.

Clearly, a major catalyst for friendships is geographic proximity. However, there must also exist other catalysts that create long-range contacts, which are the reason for the small-world character of social graphs. As in real-world, common interests are likely to cause friendship links also in the virtual world. Therefore, it is natural to expect that besides the geography layer, a second important layer in *LiveJournal* would be the *interest layer* (recall Figure 3.1).

[3]Source: `http://en.wikipedia.org/wiki/List_of_social_networking_websites`

Crawled nodes	250K
US nodes w/ geo. info.	88K
US edges	202K
Interests	673K

Table 3.1: Input data sample.

The attributes of the input data sample that we were able to crawl and use in our experiments are summarized in Table 3.1. Note that from 250K crawled users, only about 190K have indicated their country of residence, being 88K of them from the US. We will refer to the subset of US residents with known country and city of residence (i.e. known coordinates) as *US subset*. This data was crawled in 2007.

3.3.1 Evidence for the Layer-Model

Assuming the layer model holds in the case of *LiveJournal*, we can conjecture that links in the geographic layer are typically shorter than links in the interest layer. Links that purely belong to the interest layer should have random length. Links in the geographic layer, on the other hand, should show a trend toward short links, as the corresponding friendships are supposed to be caused by geographic proximity.

Using the interest information in *LiveJournal* together with the geographic coordinates in the *US subset*, it is possible to verify the above conjecture. One distance unit in our experiment corresponds to one degree in latitude. For simplicity we assumed that this space is a Euclidean plane for the area of the United States.

We define a *geo-edge* to be a friendship link between two users that do not share any interests. An *interest-edge*, on the other hand, is a friendship link between users that share at least one common interest but that does *not* "close" a triangle with two *geo-edges*, i.e., two users linked by an *interest-edge* do not have any common friends in their (assumed) physical proximity. If they have such a friend, the link is supposed to lie in the intersection between the *geography* and the *interest layer*. In our analysis, we do not take such "intersection" links into account. According to this definition, roughly 40% of the edges in the *US subset* are interest edges.

For each edge in the friendship graph, we counted the number of common interests of the two corresponding users. It can be seen in Figure 3.5 that the average length of a friendship link increases as the number of common interests increases until a threshold of approximately 5 and then stabilizes. Given that *interest-edges* present an almost 20% increase in average geographic length, we can deduce that such links are less influenced by location

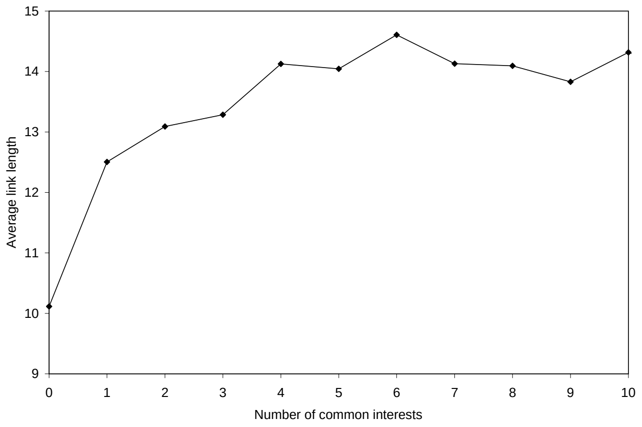


Figure 3.5: *LiveJournal*: The average geographic length of a friendship link drastically increases when there is a common interest.

of residence (i.e. geographic proximity) and can thus be considered part of the *interest layer* of the friendship graph. This observation confirms our expectation and thereby supports the layer model.

3.3.2 Location Estimation

In this section we propose an application for the properties exposed above. We have implemented a simple location estimation application to prove the concept. The experiment was performed on the *US sample*. We assume that the location of a node can be estimated by looking at the locations of its neighbors. We basically estimate the position of a node as the center of mass of its neighbors. In our case, nodes are people, and locations are their cities of residence. The rationale behind the estimation idea is that a person with many friends in, say, New York (and close-by cities) is likely to live in New York him/herself, too.

The location estimation was performed on the original friendship graph and compared to a preprocessed graph, from which all *interest-edges* have been removed. For evaluation, we measure the deviation of our estimated locations to the true locations (see Figure 3.6). To ensure independence of the single measurements, we have taken care that no node affects more than one location estimation. To still get a sufficiently large number of samples, the estimation is not based on all the neighbors of node, but only on a random subset of 5 neighbors. Nodes with less than 5 neighbors have been

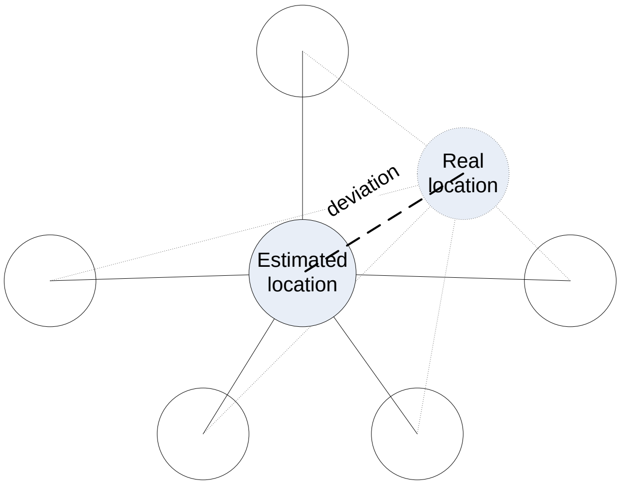


Figure 3.6: *LiveJournal*: The estimated position of a node is the center of mass of its friends.

ignored. Figure 3.7 plots the mean and median values for the deviations on the original as well as the processed friendship graphs. In both cases a random sample of 1000 nodes has been considered. Our results show that the preprocessed graph results in an average 8.30-unit location deviation, as opposed to a 9.04-unit deviation in the original “mixed layer” graph. The corresponding medians are 4.99 (preprocessed graph) versus 6.57 (“mixed layer” graph). We can conclude that the preprocessing step slightly improves the location estimation. To measure the statistical significance, we have conducted a Wilcoxon rank sum test.[4] The test rejects the null-hypothesis of equal medians at the $\alpha = 0.01$ level, and thus indicates statistical significance of the lower median observed in the preprocessed graph.

3.4 Conclusion

In this chapter we have investigated the structure of two complex networks that result from the Web 2.0 context. In particular, we have given evidence for hidden layers and hierarchies in such networks as conjectured by recently

[4]The Wilcoxon rank sum test is a non-parametric test that decides whether two data series are likely to stem from the same distribution with the same median. The departures from the null hypothesis (both data series originate from the same distribution) that the Wilcoxon test tries to detect are location shifts.

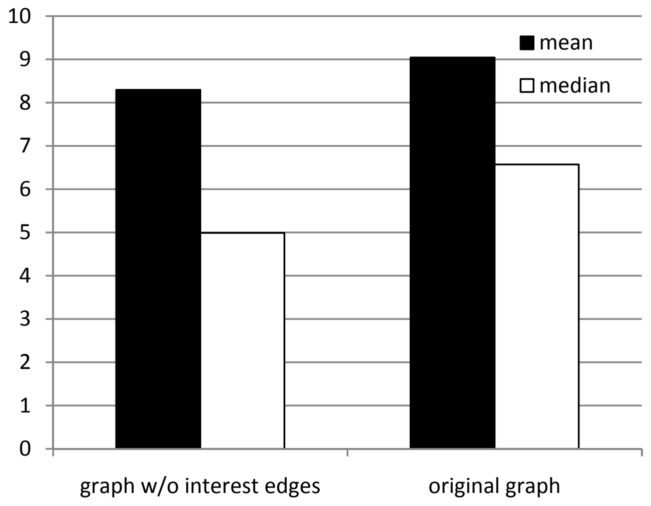


Figure 3.7: *LiveJournal*: Location estimation is more accurate if only *geo-edges* are used.

proposed network models. Moreover, we have shown how the knowledge about these properties can help to solve certain tasks more efficiently. While we only presented two exemplary tasks, namely routing and location estimation, we believe that the basic findings generalize to a variety of contexts. In fact, in Chapter 8, we will show another example in which the separation of layers as a preprocessing step improves the performance of a data mining algorithm.

Chapter 4

Recommending Contacts by Social Ties

Social services have experienced a tremendous success in the past years. Sites such as MySpace, Facebook, or LinkedIn have steadily been growing and became ubiquitous on the Internet. Nowadays, many people can no longer imagine a life without such applications. Interestingly, similar services in the mobile domain have only recently started to become successful. Different reasons are said to be responsible for the lag. Restricted input and output capabilities of mobile devices have inhibited the spreading of a "Mobile Web 2.0". Moreover, inappropriate communication technologies that offer low data rates at high prices are often seen as an important reason for the low acceptance of the corresponding services. Recent developments, however, mitigate these issues. The mobile infrastructure is migrating to a packet switched all-IP network. Wireless broadband technology is constantly improving, and 3G coverage is spreading rapidly. In addition, in many countries, a shift from time to data centric and even flat rate pricing models can be observed, which allows for permanent connectivity at low cost. Finally, state-of-the-art mobile phones, such as the iPhone and Android based devices, have literally revolutionized the user interface side. Moreover, rich APIs and transparent application distribution channels are major drivers for innovation. Considering these technological advances, it seems that the potential of mobile devices can finally be fully exploited. Thus, we expect that more and more social services will expand their reach into the mobile domain. In fact, specific mobile versions of well-known platforms such as MySpace[1],

[1] `m.myspace.com`

LinkedIn[2], and Facebook[3] are enjoying an enormous popularity. Following this trend, the microblogging service Twitter[4] has grown to one of the most relevant communication platforms at the time of writing.

Twitter has always been designed for use in the mobile domain. It thus well fits the mobile context, in contrast to most of the currently available services that do not exploit the full potential of the underlying technology. In particular, most services merely try to copy successful concepts from their desktop counterparts. However, the usage patterns of mobile phones significantly differ from those in a wired environment. It is thus crucial that mobile services take the special characteristics of the environment – such as mobility, ubiquity, and permanent reachability – into account.

The wired telephony network once lowered the communication delay and made spatial distance between communicating parties negligible. Today's mobile infrastructure continues this trend: We can reach everyone everywhere all the time – instantly. As a result, the differences between personal and remote communication patterns are diminishing. Thus, the 1:1 conversation scheme known from traditional remote communication will more and more be complemented with group interaction, much as when socializing in the real world. Future-oriented mobile social services thus have to adequately address the related issues. In particular, they should facilitate group communication and offer support for collaboration. As we will see in the next section, several ongoing research activities are devoted to this field.

One important aspect falls short in these activities: The initial group formation process. A major focus of our work thus lies on the fast and convenient initialization of group interaction on mobile devices. In a survey we identify relevant requirements to enhance group communication on mobile devices. Based on the findings, we then propose a contact recommendation algorithm which allows to quickly group people in order to contact them at once. Moreover, we outline mechanisms that simplify the interaction within such a group after its initial formation. The most important components have been implemented in *Cluestr*, a proof-of-concept application running on Windows Mobile devices. Based on this application, we have conducted a preliminary user experiment. The experiment makes use of real-world data extracted from Facebook to demonstrate the suitability of the proposed methods. In particular, we demonstrate that relevant community information can accurately be deduced from this data. Moreover, we show that, when using our recommendation engine as opposed to traditional list-based approaches, in realistic scenarios significant time savings can be achieved during group establishment.

[2]`m.linkedin.com`

[3]`m.facebook.com`

[4]`www.twitter.com`

4.1 Related Work

Traditional online social networking platforms provide only marginal functionality for enhanced communication on mobile devices. By contrast, truly mobile social services take people's mobile behavior patterns into account and offer functionality to enhance communication. Services like TXTMob [48], Jaiku[5], or Twitter implement a broadcasting system to which friends can subscribe in order to receive message updates. FriendFeed[6] enables friends to comment on such messages and thereby extends this approach. ContextContact [90] and Swarm [30] are designed to enhance communication within a large group including all of a user's contacts.

Communication in the mobile space is often used to plan, schedule, and reflect on group activities. As stated in [55], current mobile phones are not designed to support such behavior in a group context. The authors present the idea of creating privately shared group spaces on mobile devices where each group is able to communicate and collaborate in order to overcome this lack of functionality. Similar ideas have been followed by the Slam system [18]. Here, the focus lies on communication within small groups, such as class mates, co-workers, and family members. An ad hoc way to initialize collaboration, mediated through personal mobile agents, is proposed by [33].

Most mobile social applications provide some degree of support for collecting contextual information and possess functionality to publish or share this information with other members of a group. There exist plenty of applications that offer such context sharing: Some systems disclose information about their users' presence [90, 112], others about location [89, 110], motion [11] and proximity [96].

Besides group-based communication, some services provide functionality to collaboratively solve tasks using mobile devices. An example is Doodle[7], which offers a polling service for multi-party negotiation.

All approaches have in common that the group formation has to be done manually. The systems are not able to automatically deduce community affiliation from user behavior.

We next present a survey which indicates that current group establishment mechanisms do not well agree with the users needs and that more efficient methods are demanded.

[5]www.jaiku.com

[6]www.friendfeed.com

[7]m.doodle.com

4.2 Survey

The previously mentioned technological and economical developments ask for new concepts and ideas regarding mobile communication. To shed light on the usage patterns of today's mobile phones, we set up an online survey. It focuses on aspects related to group communication and collaboration in mobile settings and investigates to what extent social networking services could contribute in this context. A total of 342 people from Europe participated in the survey. The outcome can be summarized as follows:

- Most participants agreed on the fact that their contacts stored in the mobile phone's address book can be grouped into communities such as 'university colleagues', 'coworkers', 'family', 'friends' etc., and that communication often occurs among the members of a certain community simultaneously. However, although today's mobile phones allow to group contacts, this functionality is hardly used. Only 16% use the built-in grouping function to assign contacts to groups.

- 68% of all participants use the feature to send text messages (SMS) to multiple receivers. It is being used for different tasks including holiday greetings, invitations, scheduling meetings, event organization and polls.

- The conference call feature is barely used in daily life. 11% claim to use it on a non-regular basis. Only one person claimed to use it regularly. The main purpose for using this feature are business meetings.

In particular, we conclude that:

- The mobile phone is often used to organize and coordinate activities among multiple people, such as to discuss how to spend the evening, or to decide about a meeting point and time.

- Group communication is often performed with members of a community existing in real life, such as members of a sport team, coworkers, class mates or family.

- Existing features for group maintenance and group communication are rarely used. Rather, most people manually (re-)establish groups, when they, for example, want to send a message to multiple receivers.

Today's mobile communication alternatives do not cope with this social behavior. Text messages can be sent to multiple receivers simultaneously.

However, the underlying system is limited with respect to group communication. In particular, text messaging only offers a 1:N way of communication. For efficient group communication, however, an N:N solution is required.

The conference call feature included in the majority of state-of-the-art mobile phones offers such functionality. However, it is restricted to voice communication and only works in synchronous mode, i.e., all the participants have to concurrently be present (at the phone) in order to receive the information. E-mail does not suffer from the outlined problems. However, its popularity in the mobile context is still low in many regions. Reasons are, presumably, a insufficient integration in current devices, together with the limited input capabilities of these devices.

Finally, only a minority of users uses the built in feature to maintain groups of contacts. A major reason for the low acceptance is presumably the high dynamics involved. People join and leave communities (e.g. a sport team), which implies tedious work to keep the group information up-to-date. A main contribution of our work is an interface that simplifies group formation without the need for permanent maintenance.

4.3 Cluestr

In the following, we outline a service which focuses on group communication and collaboration, and addresses the major findings of the survey. The service is named *Cluestr*, a combination of *cluster* and *clue*. Cluestr should be seen as a proof-of-concept implementation and an evaluation environment. It is not our goal to promote yet another mobile social application. The main ideas and concepts addressed here can easily be integrated into existing systems.

4.3.1 Vision

Cluestr is designed to enhance communication among members of a group. In particular it helps to lower the effort to organize, manage and coordinate group activities, such as visiting a cinema, which requires a group of friends to agree on a movie, meeting point and time. A further enhancement is the incorporation of collaboration capabilities that can be used among members of a group. The following illustrative example sketches a typical situation where Cluestr is useful:

> Every Saturday, a local football team has a game. Using Cluestr, the team captain can initiate a group and invite all team mates as participants. On a billboard, team mates can then inform the others whether they will join the game or not and discuss about the meeting point. Using a poll function, they can vote for

> the person who has to be the chauffeur and drive to the game. Using a ToDo list, the team manages the logistics for the BBQ afterwards. Everyone can tick what he will contribute to the buffet.

The strength of the Cluestr service lies in its support for group communication and collaboration in general, and in a novel approach to initiate groups in particular. The participants of such groups can profit from intra-group communication and collaboration tools, such as a thread-like billboard, where everyone can post and read messages. In addition, a poll through which participants can vote, and a ToDo list that allows participants to add elements or mark them as accomplished is available for each individual group. All this functionality is well known from other contexts. However, the single features only become useful when the underlying group is known. As mentioned before, there is little support for group initialization and/or maintenance in current mobile devices. In the following we thus focus on the group initialization process.

4.3.2 Need for an Efficient Group Initialization

As discussed, one of our main contributions is an efficient method to establish a group. Groups are initiated by one person (initiator), who is then able to invite a set of contacts to participate.

Due to interface constraints, inviting contacts should be kept as simple and intuitive as possible. Selecting participants from a traditional, alphabetically ordered contact list is inefficient since the required contacts are in general randomly distributed over the whole range. Cluestr offers a more efficient method for finding the desired contacts. We present a contact recommendation engine able to support the initiator by suggesting suited contacts for invitation. For illustration, we give a simple use case:

> If the initiator of a new group invites contact A, the engine proposes that contacts B, D and E might also fit into this group but not contact C. The initiator decides to invite E as well. With this additional information, the engine understands that the initiator is, say, not interested in D and hence only recommends B.

In addition to time-saving, contact recommendation helps to remind the initiator of contacts that he/she otherwise might have forgotten, or that he/she was even not aware of.

4.3.3 Concept of the Contact Recommendation

Our recommendation algorithm is based on the existence of social dimensions (recall Chapter 3), which is also reflected in the outcome of our survey: The initiator belongs to different communities and often communicates with several members of a community simultaneously. The basic idea is to take advantage of existing community structures in the following way: Whenever a user wants to initiate some form of group interaction, he/she initiates a new group and starts by selecting a first contact. Next, the engine proposes a list of people that share one or several communities with the selected contact. This list is sorted by relevance, which is given by the number of shared communities. The user can now select a next contact from this list, which in turn gets repopulated with the entries best matching both selected contacts. This process is repeated until the group is complete.

One could think of different approaches to extract the required community affiliations of the initiator's contacts:

- Tagging of contacts and manual grouping
- Semantic analysis of the communication content
- Communication pattern analysis
- Social graph topology

Tagging and manual grouping of contacts implies a large maintenance effort by a user, which is undesired. Moreover, this feature is already implemented in many mobile phones but hardly used according to our survey. We want our recommendation engine to be able to recommend relevant contacts with the least possible user effort.

Analyzing the content of the communication to then group contacts around topics might be another approach. This idea is followed in [19, 31, 42]. However, a lot of mobile communication is voice based. Gaining relevant content information using voice recognition is hard to achieve on mobile devices.

One could also imagine that community affiliations are deducible by observing the stream of communication on the user's device. First experiments we have conducted in this direction, however, indicate that solely analyzing communication patterns does not allow to estimate community structure with sufficient accuracy. It remains open, though, whether the required accuracy could be reached if further contextual cues were included.

The approach we pursued is based on social network analysis. In a social networking service, users link to each other to indicate relationships. This leads to a network in which related users are connected through ties. We designed our recommendation engine to require only this network information

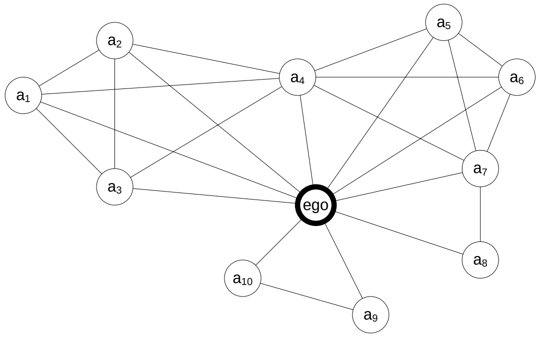


Figure 4.1: Visualization of a sample ego-graph

to find community affiliations of contacts and use this information to generate recommendations.

4.4 Contact Recommendation

The previously introduced concept of contact recommendation requires the engine to know:

- The different communities the initiator belongs to.
- Which of the initiator's contacts belongs to which communities.

As mentioned before, this information is extracted from the social network stored on the Cluestr server.[8] Before describing the extraction process in detail, we introduce some notations.

4.4.1 Notations

Under a *social network* (or *social graph*) we understand a graph $G = G(V, E)$, where the set V of vertices represents users (or contacts), and the set E of edges denotes links (or friendships) between these users. An *ego-graph* (or *ego-centric graph*) is a special form of a social graph. It consists of a user of interest (*ego*) and his/her direct neighbors (*alters* a_i).[9] A sample ego-graph is illustrated in Figure 4.1.

[8] Observe that all the presented concepts could also be based on an existing social networking service (such as Facebook).

[9] The notation (ego, alter) is taken from [123].

A *cluster* is a subset of vertices of a social graph that is highly connected. In particular, we assume that the density of edges within a cluster (intra-cluster edges) is larger than the density of edges connecting vertices from inside the cluster to vertices outside of the cluster. A *clustering algorithm* seeks to partition a graph into a set C of clusters c_i. The resulting partition is also called the *clustering* of a graph. If the resulting clusters do not need to be disjoint, the clustering is said to consist of *overlapping clusters* (i.e. one vertex can belong to several clusters).

As we will see, clusters are supposed to reflect real community structure. Throughout this section, we will refer to a *cluster* c as a group of contacts, as identified by our clustering algorithm. By contrast, a *community* o refers to a group of contacts as identified by a user. Similarly as we denote the set of all clusters by C, we refer to the set of all communities by O.

Modularity is a well known measure to estimate the quality of a clustering. It was introduced by Newman [85]. Basically, the modularity is defined to be the fraction of edges that fall within the given clusters minus the expected fraction of such edges, if edges were distributed at random (in a graph with a given number of edges and vertices). The original definition of modularity is only valid for disjoint clusters. However, Nicosia et al. [88] have recently proposed a variant which is also usable with overlapping clusters. We refer to this second definition when using the term modularity.

4.4.2 Community Detection

Figure 4.2 shows the topology of a real ego-graph retrieved from Facebook. The example graph exhibits a characteristic structure, which is common to most real-world ego-graphs. There are groups of alters that are densely connected, but only sparsely interlinked to the rest of the network. These dense regions exist due to the social characteristics of communities. The members of a community typically know each other, and thus form densely connected subgraphs. On the other side, members of one community do often not know members of other communities, resulting in a sparse interlinkage.

As mentioned before, densely connected regions can be extracted from a graph using a clustering algorithm. Graph clustering is an active area of research. A wide range of clustering algorithms have been proposed, including [15, 39, 84, 85, 86]. Due to the outlined correlation between interlinkage and communities, we expect that the clustering generated by a sophisticated algorithm well reflects the actual community structure. A careful look at Figure 4.2 reveals that some of the alters belong to more than one community. A clustering algorithm which is supposed to extract the actual communities thus needs to be able to deal with overlapping clusters. An algorithm that fulfills this property is the CONGA algorithm [40]. It extends the widely

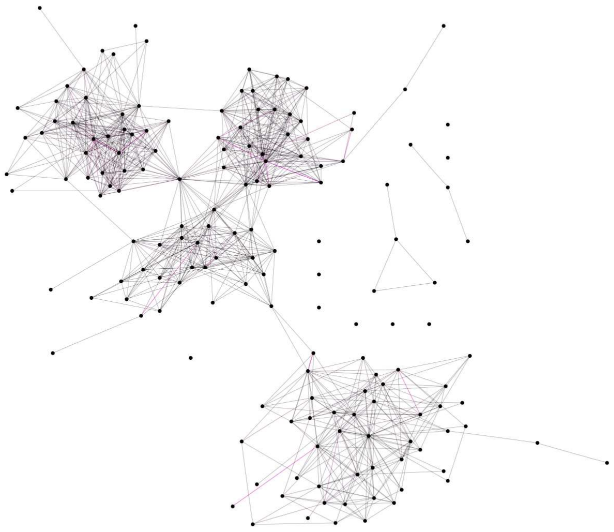


Figure 4.2: A real ego-graph: The characteristic community structure is clearly visible. The central user (ego) as well as the ego-alter ties are not displayed for simplicity.

used Girvan and Newman algorithm [39, 86], which can only retrieve disjoint clusters.

CONGA belongs to the class of divisive hierarchical clustering algorithms. That is, it starts with a single cluster containing all vertices and, by subsequent partitioning, produces an ever finer grained hierarchy of clusters. The partition process can be repeated until all nodes form single node clusters, which are the leaves of the hierarchy. To explain the algorithm in more detail, we use the notion of *edge betweenness* and *split betweenness*, according to [40]:

- *Edge betweenness*: The betweenness of edge e is defined as the number of shortest paths, between all pairs of vertices, that pass along e. A high betweenness means that the edge acts as a bottleneck between a large number of vertex pairs and suggests that it is connecting different clusters.

- *Split betweenness*: A vertex v can be *split* by partitioning the set of its neighbors into two disjoint sets s_1 and s_2. Then, v is replaced by two virtual nodes v_1 and v_2 that are connected by an edge e_v. Moreover, each node in the set s_i is connected to v_i. The *split betweenness* is defined as the maximum edge betweenness of e_v among all possible partitions into s_1 and s_2. Since the number of possible partitions can be high (roughly $2^{\delta-1}$, where δ is the degree of node v), an approximation algorithm to calculate split betweenness is used by [40].

The CONGA algorithm can then be described as follows:

1. Calculate edge betweenness of edges and split betweenness of nodes.

2. Remove the edge with maximum edge betweenness or split the node with maximum split betweenness, if higher.

3. Recalculate the edge and split betweenness values.

4. Repeat from Step 2 until no edges remain.

In [40], the time complexity of the CONGA algorithm is shown to be $O(m^3)$, where m is the number of edges in the graph. Gregory has later proposed a modified version of the algorithm (named CONGO) to reduce the time complexity [41]. However, we found that performance is not a key issue, and, as it provides more accurate results, rely on the original CONGA algorithm for ego-graph clustering.

4.4.3 Recommendation

As discussed before, we assume that most group interaction takes place within communities. Therefore, the chance that the initiator of a group wants to invite several members of the same community is high. The automatic detection of dense structures (clusters) in the initiator's ego-graph approximates the real-life communities like class mates, work colleges, family members, and so on.

The recommendation process for the initiation of a new group thus works as follows:

1. Detect hidden community structures by clustering the initiator's ego-graph. Each contact is assigned to at least one cluster.

2. Present an alphabetically sorted list from which the initiator can choose a first contact.

3. Rank contacts based on the cluster affiliations of the previously selected contacts.

4. Recommend the best ranked contacts to the initiator.

5. Continue with Step 3 after the initiator has selected a next contact to invite, or terminate if the group is complete.

The idea of the ranking function in Step 3 is to rank contacts high that share many clusters with the already invited persons. In particular, each cluster is weighted with the number of occurrences within the already selected contacts. Each of the remaining contacts is then scored by the sum of all the clusters it resides in (see Figure 4.3). If the list from Step 4 does not contain any relevant contacts, the user can switch back to an alphabetical list to continue the invitation process. Often, the recommendation algorithm recovers in the next iteration, such that an alphabetical list has to be consulted only rarely. Assume, for example, the user wants to invite both, team mates as well as co-workers, to a birthday party. If these communities – and the corresponding clusters – are entirely disjoint, the recommendation process will work fine for the first cluster, then require the consultation of the alphabetically ordered list, and afterwards work fine again for the second cluster.

4.5 Evaluation

We have investigated different aspects of the proposed contact recommendation algorithm, namely:

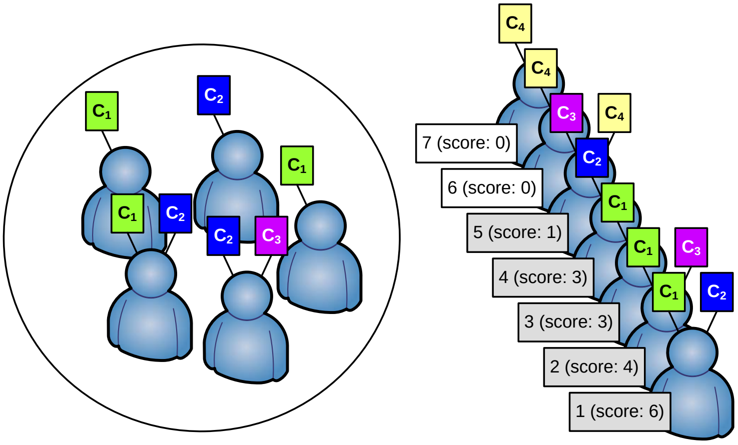


Figure 4.3: Contacts are rated according to their cluster affiliations. For each occurrence of a cluster within the already invited group members (left), a contact's score increases by 1. For the top ranked user (that belongs to clusters 1 and 2), for example, this amounts to a total score of 6.

- *Clustering accuracy*: How well do the clusters generated by the clustering algorithm reflect the social communities identified by the user.
- *Advantage of recommendation*: How much time can be saved if a group is created by using our recommendation algorithm rather than by selecting contacts from an alphabetical list.
- *Effect of sparsity*: How well does the proposed concept work if data is spare, e.g. due to a low number of registered users, or due to missing friendship information in contact books.

Evaluating an algorithm on real social graphs is challenging. It is impossible to characterize the performance of the algorithm without knowledge of the correct community structure, the *ground truth*. For proper analysis, therefore, not only information about the social graph but also about existing communities is required. Such information can best be retrieved by subject questioning.

Extracting the ego-graph from real mobile phone contact books is infeasible.[10] We thus decided to rely on Facebook data. Facebook is the presumably

[10]Extracting the entire ego-graph would require to read the contact books of all contacts of ego.

most complete and well established online social networking service. The underlying social network is thus likely to exhibit similar characteristics as the social network defined by the contact information in people's mobile phone address books.

Due to the lacking ground truth, available data sets as presented in [67] cannot be used for evaluation. For our experiments, we thus extracted the ego-graphs of four subjects (two male, two female) from Facebook. Moreover, we asked these subjects to group their (Facebook) friends into an arbitrary number of communities (such as coworkers, team mates, family members, etc.). The resulting four datasets form the ground truth for our analysis. Each of the datasets contains between 59 and 151 contacts, and was assigned between 4 and 7 communities by the corresponding subject.

4.5.1 Evaluation Measures

In the context of classification tasks, *precision* and *recall* is a widely used evaluation concept. The *precision* for a class denotes the number of items correctly identified as belonging to the class (i.e. true positives) divided by the total number of elements classified as belonging to the class (i.e. the sum of true positives and false positives). Similarly, *recall* is given by dividing the number of true positives by the total number of items that, according to the ground truth, really belong to the class (i.e. the sum of true positives and false negatives).

In our setting, we are interested in the accuracy of the extracted *clusters* with respect to the *communities* identified by our subjects. Above measures thus become to:

- Precision of cluster i with respect to community j:

$$P_{i,j} = \frac{|\{\text{contacts in } c_i\} \cap \{\text{contacts in } o_j\}|}{|\{\text{contacts in } c_i\}|}$$

- Recall of cluster i with respect to community j:

$$R_{i,j} = \frac{|\{\text{contacts in } c_i\} \cap \{\text{contacts in } o_j\}|}{|\{\text{contacts in } o_j\}|}$$

These two values are often combined to form a single evaluation measure, called *F-measure*, which is the harmonic mean of recall and precision:

$$F_{i,j} = 2 \cdot \frac{P_{i,j} \cdot R_{i,j}}{P_{i,j} + R_{i,j}}$$

Observe that the assignment of a cluster c_i to a community o_j is not known beforehand. We thus chose the o_j that maximizes the *F-measure* $F_{i,j}$ as the representative community $o_{j^*(i)}$ of c_i, i.e., $j^*(i) = \text{argmax}_j\, F_{i,j}$. The corresponding measures for the entire classification task can then be defined as follows:

- *Precision:*
$$P = \frac{1}{|C|}\sum_{i=1}^{|C|} P_{i,j^*(i)}$$
- *Recall:*
$$R = \frac{1}{|C|}\sum_{i=1}^{|C|} R_{i,j^*(i)}$$
- *F-Measure:*
$$F = \frac{1}{|C|}\sum_{i=1}^{|C|} F_{i,j^*(i)}$$

4.5.2 Clustering Accuracy

The contact recommendation algorithm can only work if the ego-graph decomposition of the clustering algorithm well agrees with the intuition of the user. The CONGA algorithm has shown to perform well in various settings [40, 41]. Nevertheless it remains to be shown that the graph structure and the resulting clustering well enough represent the perception of a user about how to naturally group his/her contacts.

We therefore investigate the accuracy and applicability of CONGA for the partition of an ego-graph into communities. First, a suitable termination criteria for the divisive clustering process is required, which leads to a well defined number of – possibly overlapping – clusters. As pointed out by Gregory [41], modularity can, but does not need to be an appropriate criteria for this purpose. Our first experiment thus investigates whether modularity maximization also maximizes the F-measure. We clustered each ego-graph until singleton clusters remained. At each *stage*, i.e. for each number of clusters, recall, precision and the resulting F-measure were calculated. A plot of clustering stages 4 to 50 is given in Figure 4.4.[11] The vertical line indicates the clustering stage at which modularity is maximized. In all but one case (Subject 4), the clustering stage that optimizes modularity agrees with the stage that maximizes the F-measure. Moreover, for Subject 4, it differs by only one cluster. This good alignment indicates that modularity is a well-suited optimization criteria for clustering ego-graphs.

[11]The evaluation starts as stage 4, since the used algorithm implementation did not allow to create clusterings with fewer than 4 clusters.

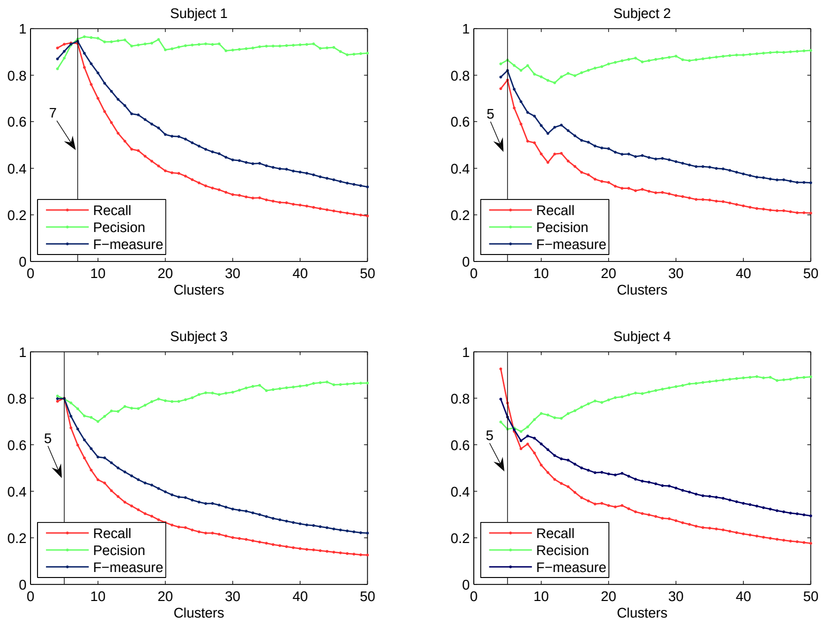


Figure 4.4: Comparison of the accuracy of clustering the subject's ego graph at different stages. The vertical line indicates the stage with maximal modularity.

While we have seen that modularity is a good criteria to maximize the F-measure, it is still not clear whether the resulting clusters well reflect the user's perception. In particular, the number of clusters detected by the algorithm (at maximum modularity) might greatly differ from the amount of communities identified by a subject. Table 4.1 compares the number of manually defined communities to the number of clusters that maximize modularity and F-measure, respectively.

The fact that these numbers all lie in the same range indicates that the modularity maximizing CONGA clustering well reflects the users' intuition concerning communities. However, we did not reach perfect correspondence. To understand the reasons for the differences, we asked the subjects to name their communities and also give names to clusters recognized by the algorithm. A comparison of the outcome facilitates a qualitative interpretation. Basically two effects caused the number of clusters to differ:

	#Communities	#Clusters (max Mod)	#Clusters (max F-measure)
Subject 1	7	7	7
Subject 2	7	5	5
Subject 3	4	5	5
Subject 4	7	5	4

Table 4.1: Comparison of the number of clusters resulting from subject questioning and CONGA.

- Combining two independent groups of friends, which are barely interlinked, into one community. Such communities are detected as different clusters by the algorithm. One subject, for example, put all friends she got to know during her internship abroad into one community. However, this community was recognized as two clusters: One containing flat mates she was living with, and one containing co-workers. There were no ties connecting these two groups.

- Two communities were discovered as only one when the interlinkage between friends was too high to separate them. This may happen when one community is a subset of another one. One subject, for example, put her friends from university into one group and defined a second community with friends she knows from a student organization. Members of this organization, however, go to the same university and possess friendships with other students not in the organization. This made it impossible to separate the two communities. As a consequence only one cluster was detected.

To quantify the accuracy of the clustering, we have applied the aforementioned quality measures, namely precision, recall and F-measure. Table 4.2 summarizes the results. The high F-measure values (average: 82%) achieved by automated clustering are promising especially when taking the two above stated causes for non-congruency into account. We conclude that a user's ego-graph contains sufficient information to realize a contact recommendation system, and that this information can be efficiently extracted by the CONGA algorithm.

4.5.3 Recommendation Engine

In the previous section, we have shown that clustering an ego-graph using CONGA at maximized modularity results in an adequate estimation of communities and their affiliated contacts. Next, we evaluate whether recommendation based on graph clustering is able to improve the group initialization

Subject	Recall	Precision	F-measure
Subject 1	0.94	0.96	0.95
Subject 2	0.78	0.87	0.82
Subject 3	0.80	0.79	0.80
Subject 4	0.78	0.67	0.72
Average	0.83	0.82	0.82

Table 4.2: Clustering accuracy (recall, precision and F-measure) for each subject.

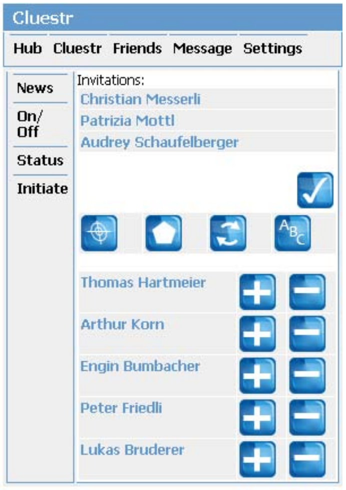
process. In particular, we are interested in the time savings a user can achieve in a realistic scenario when using our recommendation engine. For this, we asked the subjects to establish groups and send invitations to friends from their contact list using Cluestr. Three different groups had to be established according to different scenarios:

1. "Invite all members of a community of your choice for a BBQ."
2. "Invite some of your contacts from one community of your choice to watch a movie at your home."
3. "Invite a random selection of contacts for participation in a user study."

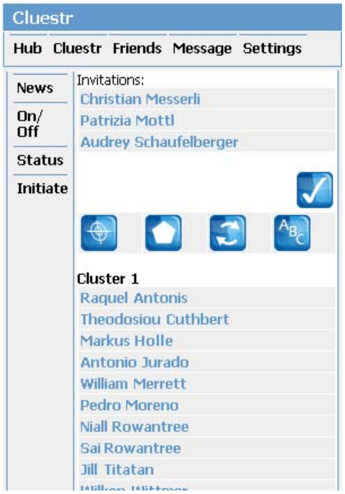
The scenarios pose increasing challenges to the recommendation engine. In the 1. Scenario, the group consists of all members of one community. In the 2. Scenario, only a part of a community's members should be invited. The 3. Scenario, finally, deals with a random sample of contacts, regardless of their community affiliation.

The experiment was performed on a mobile device (HTC Touch Cruiser) using Cluestr. The subjects' ego-graphs from Facebook were used as the underlying data set for this experiment.

Each task had to be solved following the same procedure: In a first step, the subjects were asked to write down all participants they wanted to invite. Then, these participants had to be invited using Cluestr. Each subject had to perform the invitation procedure on the device three times. First, the subject was shown a traditional *alphabetic list* of his/her friends, second, he/she had to establish the group using a *cluster view*, where friends are grouped according to the detected clusters (see Figure 4.5(b)). Third, the subject had to choose the participants using the *recommendation view*, in which the subjects could select contacts from the top 5 recommendations (see Figure 4.5(a)). During the experiment, the time required to initiate a group was measured. Figure 4.6(a) shows the average time required to select a participant in each scenario and each selection mode. Figure 4.6(b) plots the mean values over all subjects.



(a) Group initialization process: recommendation view



(b) Group initialization process: clustering view

Figure 4.5: Screenshots of Cluestr running on a Windows Mobile device.

The results show that the recommendation algorithm performs stronger, the more community-centric the group is. The more randomized the selected contacts are (in terms of community affiliation), the more challenging it gets for the recommendation engine to come up with adequate suggestions. In a completely randomized situation, as evaluated in Scenario 3, recommendation performance thus decays.

During our experiment, whenever the recommendation was inappropriate, the subject could switch to either the alphabetic or clustered list to select a next participant. The less community focused the group was, the more frequent a subject had to switch to these selection modes. However, our survey indicates that most groups are established with contacts that form a community in real life. Therefore, Scenario 1 and Scenario 2 are more realistic.

These results show that both, listing contacts according to cluster affiliation as well as recommendation based selection, results in time saving. Contacts can be found faster and in a more convenient way. In our experiment, the average selection time per user in Scenario 1 could be cut in half from 12.2s to 6s by using our recommendation engine. If only a partial, rather than an entire community needs to be selected, a reduction of 23%, from

11.5s to 8.9s, was achieved. The average size of the initated groups consisted of 19 contacts, resulting (in average) in an overall timesaving of 120s and 50s, respectively. Besides the improvement in time, the subjects also mentioned that the recommendation helps to remember all relevant participants and thus reduces the risk that somebody is unintentionally not invited.

4.5.4 Clustering Stability

As mentioned before, we assume that the ego-graphs retrieved from Facebook are well established and therefore stable. Typically, people who are registered on Facebook are already connected with most of the relevant users. Moreover, for a single user, a significant amount of friends are likely to be registered on Facebook. If a new social networking service is launched, the situation is quite different. In the beginning only few people are registered and many friendships links are missing. Our recommendation engine should also work in such environments. That is, it should yield good results even in an early phase of the service, and not require the social graph to be complete.

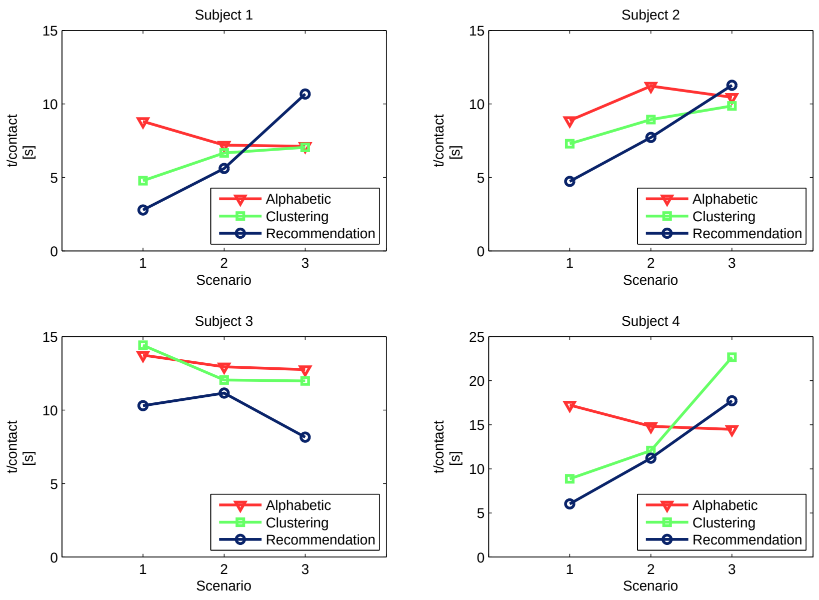
We thus also analyzed our recommendation algorithm's performance on degenerated networks, which are supposed to resemble the topological structure of a social services in an early stage. Two aspects have been investigated:

- **Missing friendships**: To reproduce the effect of missing friendships, we randomly removed links from the subjects' ego-graphs.

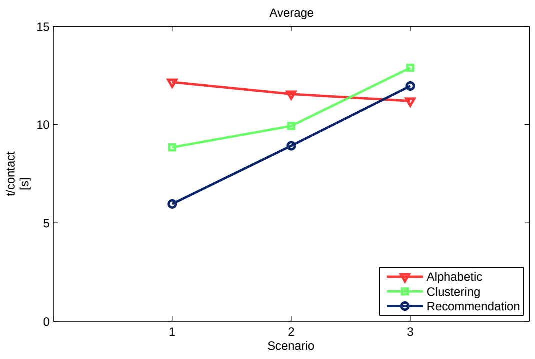
- **Missing users**: To investigate the effect of user sparsity, such as encountered shortly after launching a new services, we randomly removed nodes from the ego-graphs.

In both scenarios, the percentage of removed items (links or nodes, respectively), was increased from 0% to a total of 90% in 10% steps. After each step, the resulting graph was clustered. Possibly resulting single node clusters were ignored, as they are not relevant to the recommendation engine. To mitigate random effects, the entire procedure was repeated 30 times. For both scenarios we assessed the accuracy of the detected clusters. Moreover, we investigated the development of the F-measure, i.e., whether it remains stable or decays.

Figure 4.7(a) analyzes the effect of missing ties on the number of clusters. It plots mean and variance of the number of detected clusters at each step. The figure shows that the number of clusters increases with an increasing number of missing links. The reason for this behavior is that formerly densely connected nodes start to lose connectivity when links are removed. As a consequence, clusters fall apart into smaller pieces. Nodes do not get assigned to wrong clusters but remain together with other nodes of the same



(a) Time required to initiate a group by each subject for each scenario.



(b) Average time required to initiate a group for each scenario.

Figure 4.6: Time measurement of the group initialization experiment. For each scenario, each of the 4 subjects had to create a group using the three different selection modes: Choosing contacts from an alphabetic list, from the cluster view and based on recommendation.

community. As a consequence, the precision of the newly generated clusters remains high. The recall, however, decreases rapidly when clusters split up, which also pulls the F-measure down. These effects are illustrated in Figure 4.7.

The second experiment addresses bootstrapping issues. Shortly after launching a new social service, the number of subscribers is typically low. Nevertheless, the service has to be usable in order to attract new users. Therefore, we investigate the effect of user sparsity on the performance of the clustering accuracy. The difference to the previous experiment is that nodes rather than links are removed from the graph. Removed nodes reflect an initiator's real life contacts that are not (yet) subscribed to the service.

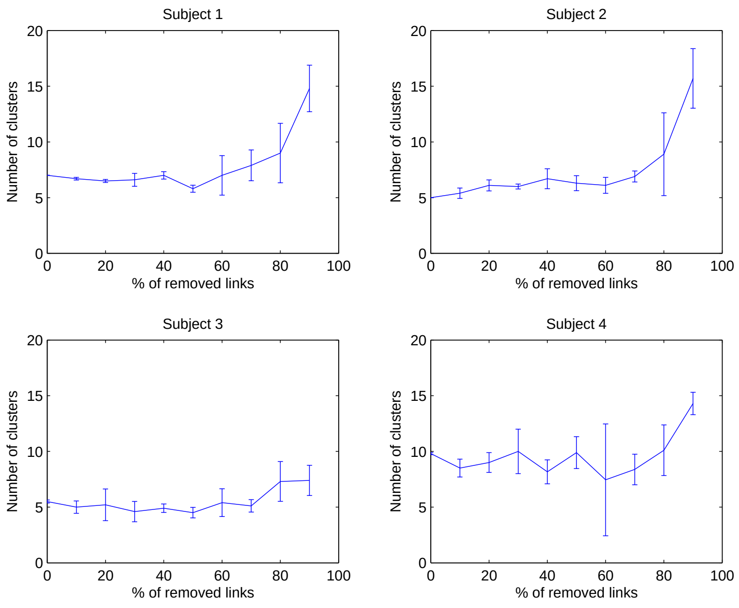
The same measurements as in the first experiment were applied and evaluated. Figure 4.8(a) shows that the more nodes are missing, the less clusters are found. The reason for the decay of the number of estimated clusters is that the clusters dissolve since their nodes are not present anymore.[12] Figure 4.8(b) shows precision, recall and F-measure values. The F-measure is high regardless of the amount of removed nodes. Although the number of clusters decreases, the high precision value indicates an accurate clustering. Apparently, missing nodes do not significantly compromise the algorithm's performance. Consequently, recommendations based on an incomplete graph are still adequate.

We conclude that only a small subset of a user's real life ego-graph is required to estimate community affiliation among these contacts. Therefore, even with a small user base, Cluestr is able to provide acceptable recommendations. This helps to overcome the bootstrapping problem. However, contacts that are not registered to the service will never be recommended.

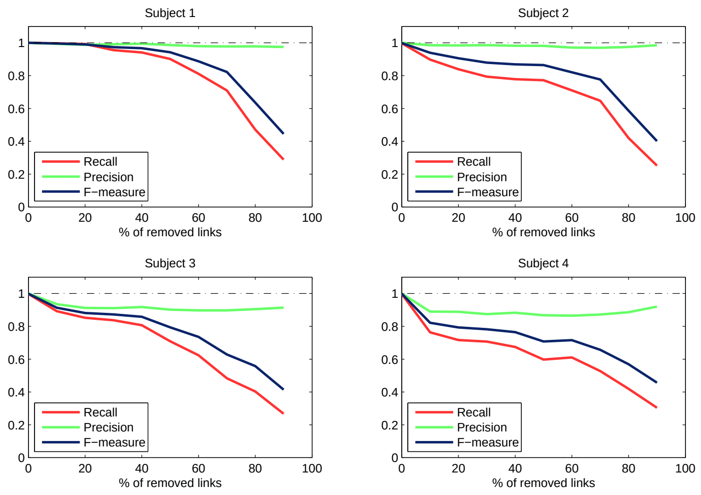
4.6 Conclusion

Group interaction is likely to play an ever bigger role in the rapidly changing world of mobile communication. Current mobile devices, however, were not designed with this kind of application in mind. A particular shortcoming is the lack of support for the initialization of group interaction, which is particularly cumbersome on small devices with limited input and output capabilities. Thus we have proposed a contact recommendation mechanism which is designed to ease the group initialization process. We have integrated the proposed recommendation engine into a proof-of-concept application. Preliminary experiments demonstrate the advantages of contact recommendation for group initialization. In particular, we have shown that:

[12] Recall that single node clusters have been ignored.



(a) Mean value and variance of number of detected clusters at each stage

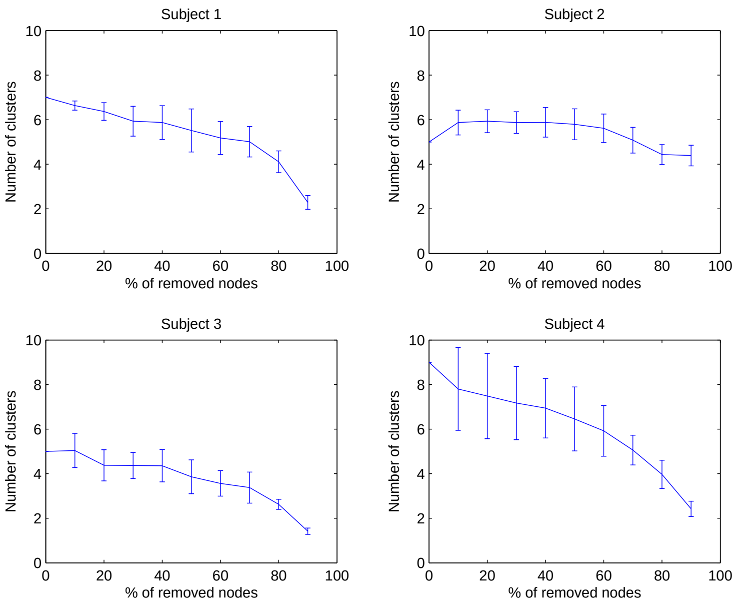


(b) Recall, precision and F-measure at each stage

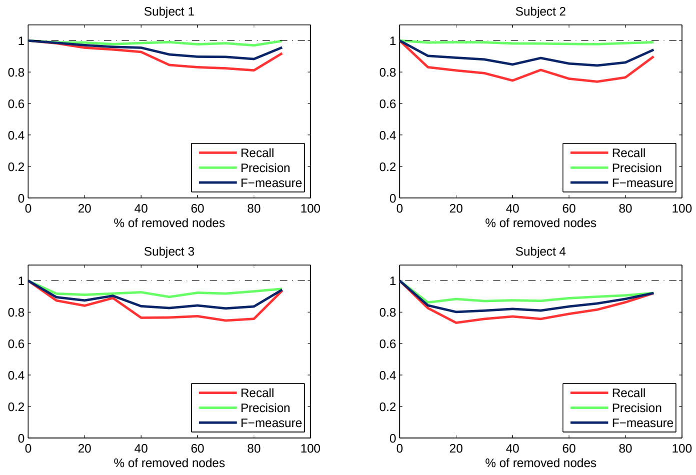
Figure 4.7: Effect of clustering an ego-graph with missing friendship links. Links were removed randomly. Each stage was evaluated 30 times.

- A user's ego-graph contains a significant amount of community information.
- This information can accurately be extracted by means of a state-of-the-art clustering algorithm.
- The extracted clusters can be used to recommend contacts that might fit into an existing group.
- This recommendation process can save a considerable amount of time when it comes to group contacts on a mobile device.

Moreover, we have demonstrated that the accuracy of the approach only moderately decreases if data gets sparse. Thus, our approach is applicable even in upcoming services that do not yet possess a large and stable user basis.



(a) Mean value and variance of number of detected clusters at each stage



(b) Recall, precision and F-measure at each stage

Figure 4.8: Effect of clustering a degenerated ego-graph with missing contacts. Nodes were removed randomly. Each stage was evaluated 30 times.

Chapter 5

Finding Friends by Social Ties

While popular social networking systems differ in many aspects, they still all build – as their name suggests – around one common component: The possibility to explore a user's social network. Apparently, people like to see who their friends' friends are, and they also like to get in contact with them, as reflected by the high clustering coefficients of social networks. Friends of friends play an important role in society. They are not only major actors in gossip and stories, but also proof extremely important in many other contexts. People are hired because their friends tell them about a friend who has a particular job opening. Headhunters of large companies nowadays try to take advantage of this fact. The friend of friend job recommendation idea is also the major feature of the social networking platform *LinkedIn*. Friends of friends are, however, not only important in the job market, but also in other businesses, and during everyday life, such as when people socialize. In fact, the high clustering coefficients of social networks indicate that friends of friends are an important ingredient of our social and emotional surrounding and that they are likely to become our (direct) friends.

5.1 Friend-of-Friend Detection

We believe that a common friend at least as well indicates a potential "match" between two persons as typical profile information of matchmaking applications, such as common interests or character traits, does. Interestingly enough, we all implicitly carry this extremely valuable information around, but do not systematically take advantage of it: Whenever two strangers meet in the street, all they have to do to figure out whether they are friends of a (common) friend, is to compare their mobile phones' contact books. We have implemented a mobile application called *VENETA* that, amongst other features, realizes exactly this idea. Whenever two *VENETA* enabled mobile phones come into Bluetooth[1] connection range, they compare their contact book entries. If none of the users appears in the other's contact book (i.e. the users are not yet friends) and they share at least one common contact, then the two users are identified as friends of a friend. Observe that such a comparison can easily be done, as the phone numbers (and similarly e-mail addresses, etc.) act as globally unique identifiers. This basic idea is illustrated in Figure 5.1. Surely, some contacts, such as the provider's help desk number, the doctor, or 911, do not reflect real friends and have to be deselected prior to the comparison. Moreover, due to different prefix formats (e.g. `+1` vs. `0011-1` vs. `001`), only the last 7 digits of the phone numbers are compared.[2] Observe that this kind of friend detection takes advantage of implicit human computer interaction [105] and thus does not require any tedious explicit actions of the user (other than installing the application).

Unfortunately, the basic idea has a relevant snag: Hardly anybody wants to broadcast his/her contact book to everybody. One might think that this problem can easily be solved by simple cryptographic hash-functions, i.e., by sending and comparing hash values instead of the actual phone numbers. However, an adversary could simply construct a lookup table containing the hashes to all the possible 10^7 phone numbers. Therefore, a more sophisticated solution is required. The next section discusses how the decentralized friend of friend finding idea can be realized in a privacy preserving way.

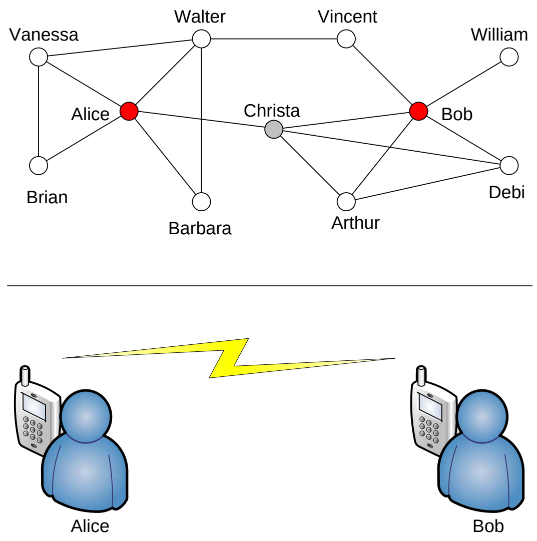
5.2 Preserving Privacy

How can two persons compare their address books without revealing the contacts (except for those that match)? More formally, two parties each own a set and they want to find the intersection of these sets without revealing the own set to the other party. This problem is known as *secure two party set intersection*, and belongs into the cryptographic area of secure multi-party computation. The goal of secure multi-party computation is to evaluate a function (or algorithm) that takes an input value of each participating party. At the end of the protocol, each participant should know the result. However, none of the participants should know more about the other participants' input values than what can be derived from the result and the own input value. Throughout this section, we will make use of the following notations:

- *Encryption*: We will denote the encryption of a message m with key κ as $E_\kappa(m)$.

[1] Any other short distance wireless connectivity technology would work as well.

[2] Note that the probability that two persons meet and own a contact with (semantically) different prefixes, but coinciding "base number" (i.e. the last 7 digits) is negligible.



Barbara Smith	1246098
Brian Lewis	6583404
Christa Doe	4320438
...	...
Vanessa Platt	7280422
Walter Boyd	8750557

Arthur V.	7092139
Christa D.	4320438
Debi B.	4457622
...	...
Vincent M.	7764982
William M.	5833210

Figure 5.1: Detecting friends of friends: Whenever the mobile phones of two strangers come into connection range, they can compare their users' contact books. If a matching entry is found, the two strangers are informed to be friends of friends. A possible social network that reflects the information from the contact books is illustrated in the top.

- *Commutative Encryption*: A commutative encryption scheme is invariant to the order in which encryption and decryption functions are applied. In particular, the following holds: $E_\alpha(E_\beta(m)) = E_\beta(E_\alpha(m))$.
- *Homomorphic Encryption*: A homomorphic encryption scheme allows to calculate $E_\kappa(m_1 + m_2)$ given $E_\kappa(m_1)$ and $E_\kappa(m_2)$.
- *Passive Adversary*: Passive adversaries try to find out something about the others' input values, but strictly follow the protocol. They are also called *semi-honest*, or *honest but curious adversaries.*
- *Active Adversary*: As opposed to passive adversaries, active (or *malicious*) adversaries do *not* necessarily follow the protocol. They can do whatever they want to compromise the other party's privacy.

It is known that any $0-1$ valued function can (theoretically) be evaluated in the secure multi-party computation model under the assumption of *passive adversaries* [128]. Unfortunately, the generic method this result is based on is computationally too expensive for most real-world problems. Moreover, real adversaries might deviate from the protocol, such that specialized solutions are required.

There are two major approaches that reduce the computation and communication complexity of two party set intersection protocols compared to generic solutions: Either an algorithm is based on a commutative encryption scheme or it exploits the power of homomorphic encryption in combination with polynomials. Both alternatives exhibit linear communication complexity. The basic construction of the second approach, as first proposed by Freedman et al. [36], is vulnerable to a simple attack in the presence of active (or malicious) adversaries. The problem has been recognized and addressed by the original paper as well as succeeding variants [36, 51, 54]. All of these fixes, however, make the protocol rather complicated. Therefore, we decided to rely on a construction based on commutative encryption. Huberman et al. [52] have proposed the following basic variant. Alice (A) owns a set $X = x_1, ..., x_N \subset V$, and Bob ($B$) owns a set $Y = y_1, ..., y_M \subset V$ (where V is the universe of all possible elements). In our case, X and Y correspond to the phone numbers in Alice's and Bob's contact books, respectively, and V is the set of all 10^7 possible phone numbers. The following protocol allows to find $X \cap Y$:

1. $A \rightarrow B$: $E_\alpha(x_1), ..., E_\alpha(x_N)$ (α randomly chosen)
2. $B \rightarrow A$: $E_\beta(y_1), ..., E_\beta(y_M)$ (β randomly chosen)
3. $A \rightarrow B$: $E_\alpha(E_\beta(y_1)), ..., E_\alpha(E_\beta(y_M))$

4. $B \rightarrow A$: $E_\beta(E_\alpha(x_1)), ..., E_\beta(E_\alpha(x_N))$

5. Both, A and B, can compare the lists from step 3 and 4. Due to the commutativity assumption, if $x_i = y_j$ then $E_\beta(E_\alpha(x_i)) = E_\alpha(E_\beta(y_j))$. Moreover, both parties know the original elements (and their order) in one of the lists, such that they can derive the matching elements (phone numbers, in our case).

Observe that this protocol does not only reveal the set intersection, but also the size of the input sets (which is not critical in our context). Agrawal et al. [1] provide a detailed analysis of the protocol. They show that, given the decisional Diffie-Hellman hypothesis (DDH) holds, the following encryption function satisfies the requirements of the protocol:

- $E_\kappa(m) = h(m)^\kappa \mod p$.
- p is strong prime, i.e. $p = 2q + 1$, with p, q prime. Clearly, p has to be large enough, such that the discrete logarithm problem is hard.
- Dom $\mathcal{E}$ consists of all quadratic residues mod p.
- $\kappa \in 1, 2, ..., q - 1$.

Agrawal et al. further assume that there exists an ideal hash function $h : V \rightarrow$ Dom $\mathcal{E}$ that maps each element $v \in V$ to a perfectly random element $d \in$ Dom $\mathcal{E}$. In our implementation, $h(\cdot)$ is a "normal" cryptographic hash function. Figure 5.2 illustrates the complete protocol for our example.

The protocol has further been analyzed with respect to malicious adversaries. Zhang and Zhao [130] as well as Li et al. [68] state two major problems of the protocol under this adversary model:

- Simply changing the input set, such as extending it by a few elements, can compromise the other party's privacy. We believe that this is not a relevant issue in our case. If somebody can extend the input set, he/she could clearly have had the corresponding contact in the contact book. Since the input set sizes are revealed, simply sending all 10^7 possible phone numbers is also not possible. In fact, for performance as well as security reasons, we restrict the input sets to contain a maximum of 300 entries.

- The protocol is not symmetric. B could decide to skip step 4 of the protocol, such that A does not know the matches, whereas B does. Zhang and Zhao refer to a cryptographic primitive that allows to "simultaneously" exchange values. Due to the additional complexity and the related performance issues, we have not implemented this idea.

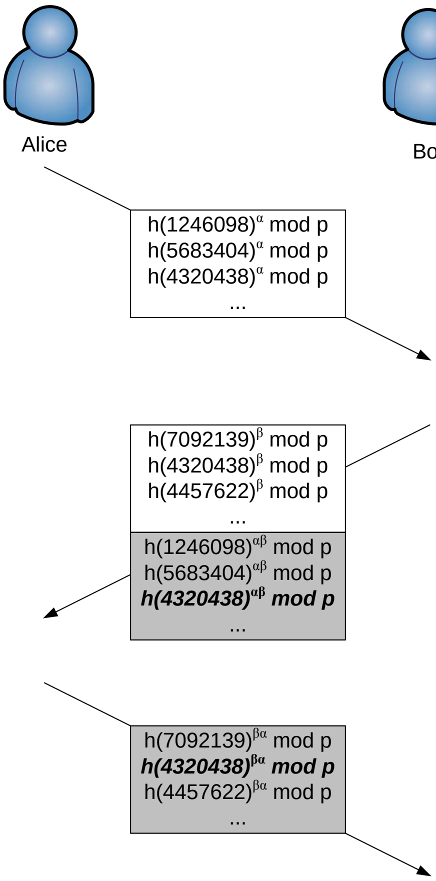


Figure 5.2: The complete protocol for our example phone numbers (recall Figure 5.1). After the last step of the protocol, both, Alice and Bob, know the shaded lists. After comparing the lists, they find that there is a matching entry, namely Christa's phone number (4320438).

> We assume that the attack cannot cause any serious damage. A victim only reveals a contact he was willing to share, without getting this information in return. Moreover, observe that such an attack is very unlikely to happen: If Alice wants to compromise Bob's privacy, she has to guess a contact she *might*[3] have in common with Bob. To be of any relevance, this contact would further require to be of delicate nature. Most likely, however, Bob would not have made such a delicate contact available to *VENETA*, such that Alice's chance of success is virtually zero.

Zhang and Zhao finally introduce an intent based adversary model and a utility function that measures the trade-off between privacy disclosure and correctness of the result. They show that the basic protocol is secure if there is a mutual interest for the information sharing to succeed, and the participants are rational (in a game theoretic sense), even if adversaries do not follow the protocol.

5.3 VENETA

A system that solely implements the contact matching idea exhibits a major bootstrap problem: If only few people use the system, the probability that two people possessing an identical contact meet is extremely low. Consequently, the contact matching functionality has to be embedded into a more comprehensive system. We have therefore developed *VENETA*[4] (available at `www.veneta-project.net`), a mobile social networking platform that addresses the outlined bootstrapping problem from two sides.

On one hand, additional infrastructureless functionality has been integrated, which decouples the system from any network provider and is thus free of charge. On the other hand, we did not want to sacrifice the possibilities of server bound features. Despite the incurred costs, we have thus implemented a central server which is able to provide benefits even for users that are not in each others immediate proximity. As this functionality is dependent on the aforementioned provider dependent billing models, all server bound features are optional, and care has been taken to keep data volumes low. The resulting platform exhibits the following features:

- *Contact matching:* (see Section 5.1).
- *Profile matching:* Besides contact matching, a traditional user profile based matching has been implemented. A search mode only considering age and gender (including a wild card) ensures a chance of success

[3]Only if she is not sure, she can gain information.

[4]The application name is derived from the latin word *veneta* (blue, sea blue), to emphasize the decentralized character (*Blue*tooth).

even for low user density. Such a trivial matching scheme is particularly important in the bootstrapping phase. We are convinced that the Japanese system *Lovegety* [53] (see Section 5.4) only had success due to this simplicity.

- *Decentralized messaging:* Often, SMS-messages are sent over short distances (e.g. at a sport event, to locate the office mate on the same floor, in an auditorium, at school, etc.). We have implemented a simple Bluetooth based messaging service that delivers messages up to 3 hops using epidemic routing. We believe that this feature is particularly appealing to young users with limited budget.

- *Server bound messaging:* Decentralized messaging is smoothly extended by centralized messaging. To keep in touch with friends even if they are not currently close-by, messages can also be delivered by the server. For privacy reasons, all data traffic is encrypted. The server acts as an intermediary to bind public keys to phone numbers (which are verified using SMS) and to distribute these keys when required.

- *User location tracking:* For devices that implement the Java Location API (JSR-179), the user location can be tracked and submitted to the server. The server can (optionally) notify users, if potential matches (according to the profile) are close-by. For privacy reasons, we do *not* allow location queries on a user basis (i.e. “where is user xy”).

We again want to stress that the system can be used independently of the server (at the expense of the server based features, of course). Figure 5.3 shows *VENETA* running on a Nokia E65 phone.

Finally, we would like to remark that VENETA could be extended by taking information from existing social networking platforms into account. Existing profile information could be imported for profile matching, and the friend of friend detection algorithm could make use of other kinds of identifiers, such as *ICQ* numbers, e-mail addresses, and so on. Such extensions might further reduce bootstrapping issues.

5.4 Related Work

Mobile social networking has been an active field of research over the past years. As a consequence, a wide variety of systems have been proposed. Many of these systems primarily rely on central infrastructure, and thus do not fall into the main focus of this work. Examples are *Plazes*, *Dodgeball*, *Jambo*, *Jaiku* or *Bluepulse*. All of them build around the location awareness concept, which is typically combined with traditional social networking functionality,

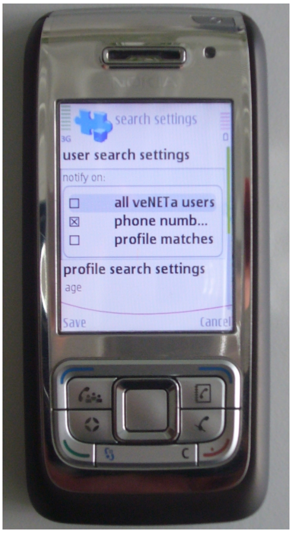


Figure 5.3: VENETA in action: The application can be configured to notify its owner whenever other VENETA users are nearby, if contact matches are found, and/or if profile matches are found.

such as the (centralized) exploration of friendship links. Some of them, such as *Jaiku* also offer some Bluetooth based features. The power of such features has been recognized by the developers of mainly decentralized applications. Commercial examples are *Nokia Sensor* [96] and *MobiLuck*, which both offer profile based matchmaking via Bluetooth. More remarkable is probably the *Lovegety* device from Japan [53]. The dedicated device exists in a "male" and "female" version, and in addition features a "mini-profile" (3 possible choices, one of which is a wild card). The device alerts the user once a potential match is nearby. Since its introduction in 1998, more than 600K of these devices have been sold. Lovegety is probably the most successful purely mobile social system so far – presumably due to its simplicity.

Another approach was followed by *Sixsense*, which relies on laptops rather than mobile phones as primary platform. It is mainly thought for "quasi-static" settings, such as in class or in a library. In addition to Bluetooth, *Sixsense* makes use of Wi-Fi to explore the neighborhood.

A purely decentralized approach from the research community is Social Net, a matchmaking application proposed by Terry et al. [113]. In this system, patterns of physical proximity are used as an indicator for a potential match. Moreover, the authors stress the benefits of a mutual friend when people are getting introduced to each other. Another remarkable contribution from the academic world is the *Social Serendipity* project [27]. As part of the project, a mobile platform has been implemented. The system makes use of Bluetooth traces to track user behavior, which enhances the – again – server based profile matching techniques present in the system. The work, however, goes beyond matchmaking. The authors emphasize the big potential of mobile social software in various settings and provide user studies that indicate the high acceptance rate of such applications.

Interestingly, for most of the server focused approaches, browsing through friendship links to find friends of friends is a central feature. Decentralized systems, however, typically lack this features and build around other, mostly location related, concepts. We believe that this is mainly due to the problems of a server independent implementation of a friendship browsing scheme. In our work, we have thus addressed these issues. By relying on *real* friends (rather than only those people participating in the system), and having the system searching (rather than the user browsing) for friends of friends, we mitigate a major problem: The lack of user density in mobile applications.

5.5 Conclusion

We believe that a major target group of social mobile applications are young people that dispose of a low budget. We have addressed this group by implementing an application that provides a wide range of functionality free of charge. In particular, we have presented a technique that seamlessly incorporates the friendship exploration functionality of traditional social networking websites into purely decentralized environments. The arising privacy issues have thereby adequately been addressed. Considering the high popularity of such features in server based systems, we believe that friend of friend detection mechanisms might become an important ingredient in upcoming mobile social applications.

Chapter 6

Concluding Remarks

Recent developments, such as the emergence of the “Web 2.0” and the mobile Internet do not only have a significant impact on social interaction, but also make a huge amount of data accessible and thereby facilitate the analysis of the resulting interaction patterns. In this chapter, we have on the one hand shed light on the structure of complex networks arising from Web 2.0 services, and on the other hand taken advantage of some relevant properties of such networks to develop new mobile social applications.

The investigation of the LiveJournal friendship, and the Wikipedia article graph has confirmed recent models that suggest the existence of hidden concepts that influence the structure of naturally grown networks. In particular, we have provided evidence that a set of hidden hierarchies affects the emergence of links in such graphs. The result is a layered structure on the one hand, and high local clustering on the other hand. We have then presented two mobile applications that try to take advantage of these properties.

Cluestr facilitates a fast group initialization based on knowledge about community affiliations. Such communities typically reflect the high local clustering caused by one or the other reason for friendships to appear, such as identical hobbies, geographic proximity, similar occupation, and so on. Observe that these properties exactly correspond to the layers (the different reasons) and hierarchies (each of the reasons can be represented as a hierarchy) identified before. The decomposition of the network along the communities facilitates the recommendation of contacts that well match a set of previously selected people. Using such recommendations, Cluster helps to significantly speed up the ad hoc selection of contacts that are addressed as a group.

The second application, *VENETA*, tries to take advantage of the fact that people that share a common friend are likely to have something in common, too. It facilitates serendipitous friend finding by automatically detecting such friends of friends. Care has been taken to preserve the privacy of the users throughout the detection process that unobtrusively runs in the background.

These examples show how the knowledge about structural properties of social behavior, measured from online data sources, can be used to build end-user applications. The next part of this thesis goes into a similar direction. However, rather than targeting at social interaction, we try to measure similarity among items. For this purpose, we also study online data. The extracted similarity information is then again used to build applications that bring direct benefits to the end-users.

Part II

Social and Semantic Similarity

Chapter 7

Introduction

With the emergence of the Internet and the world wide web, people faced a new problem, the problem of information overload. Soon, the first web search engines replaced early file archives and site directories to support people in their information search. Web search quickly grew to an important topic within the information retrieval research community, and big companies started to understand its market potential. A major breakthrough was the PageRank algorithm [91] that marks the beginning of Google's success story. Due to the extreme size of the web, traditional web search engines are nowadays extremely resource intensive and thus provided by only few large companies. However, there is a huge niche market for specific search tasks and techniques, both, commercially as well as in terms of research.

Early search engines merely looked up some keywords in some database indices and reported back the matching records. A major problem with this approach is that the users do often not precisely know what they want or they do not know how to articulate what they want. Different strategies are applied to overcome this problem. In particular, we can observe a shift from pure search engines towards a mixture of personalized search and recommendation systems. Rather than simply "retrieving what we are looking for" these systems try to help us to "discover things we are interested in". Think, for example of Amazon's online store that does not only find a requested book, but also immediately recommends other books using the "customers who bought this item also bought..." strategy (also known as item-to-item collaborative filtering [71]). Another famous example that underlines the market potential of recommender systems is the Netflix price for movie recommendation. A 1 million dollar price was offered by a DVD rental service for the first team that achieved a 10% improvement over their own predicted user ratings for movies. For almost 3 years, several thousand teams have competeted for the price until in autumn 2009 the required improvement was finally reached.

An essential ingredient in future information retrieval systems is some knowledge about the relationships among the items it helps to discover. A wide variety of signals is taken into account to understand the relatedness of objects and documents. Algorithms on the one hand examine the properties of the items themselves (content-based approach), and on the other hand take into account the behavior of the users (collaborative filtering approach), such as in the Amazon and Netflix examples. Both approaches exhibit advantages and disadvantages. While collaborative filtering is said to well correspond to the users' opinion, it is less versatile than content based approaches, and can only be applied once a sufficient amount of usage data is available. In the context of our work, context based approaches play a minor role. Rather, we make extensive use of the item-to-item collaborative filtering idea. As opposed to traditional collaborative filtering, which works on a user-item rating matrix (see, e.g. [59]), item-to-item collaborative filtering works on co-occurrence data, and is often used in conjunction with implicit information (e.g. item co-occurrences in shopping cards).

In the following, we investigate data relationships and propose retrieval approaches for two domains: Scientific conferences and music. For both domains we derive similarity measures that take human behavior into account and that are closely related to Amazon's item-to-item collaborative filtering idea. In the case of scientific conferences we come back to the layer idea presented in Chapter 3. In particular, we show that our similarity measure, derived from the authors' publication behavior, can be subdivided into multiple components, and identify a thematic and a quality layer as the major ingredients. We take advantage of this property to propose a novel conference rating algorithm. Moreover, we have implemented a conference search engine, *confsearch*[1], which fills a niche in the domain of search engines. We conclude the analysis of publication data by drawing a map of scientific conferences that gives some – not too seriously ment, and questionable – insights into the world of computer science.

We pick up the idea of a map in the music part. Using techniques from the fields of graph embedding and information retrieval, we construct two music maps that represent the similarity among songs as derived from social signals, such as the analysis of the users' listening behavior, and social tags. Essentially, songs are placed in Euclidean space such that similar songs are close to each other, and distinct songs are far apart from each other in the space. For the purpose of music retrieval, such map representation exhibits several advantages over the pure similarity information. It facilitates novel ways to create playlists, greatly simplifies the visualization of a collection,

[1]`www.confsearch.org`

and proves extremely powerful on resource restricted devices. In the end of the chapter we present a comprehensive mobile music application that bases on such a map of music. In an extensive usage study based on this application we could show that the proposed music retrieval features are highly appreciated by the community.

Chapter 8

Publication Data

Imagine your research has drifted into a field unfamiliar to you, and you do not know where to publish. In such situations it is helpful to have a better understanding of the world of computer science conferences. In the following we will explore this world from different angles. Namely, we will present a conference search application that is based on findings about conference similarity, and we will – in a playful manner – draw a map of conferences.

The starting point of our research are recent findings in the context of social networks, as outlined in Chapter 3. These findings indicate that nodes in natural graphs are interconnected for different reasons, such as common interests, close geographic distances, or family relations in case of friendship networks. Based on the researchers' social communities we will introduce a similarity measure for conferences and set up a conference graph. Similar as in friendship networks, edges in this graph are caused by different reasons – we will refer to them as the layers of our graph. Such reasons surely are area of research, but maybe also the quality, or the geographic location of the conference. In the following, we demonstrate that and how it is possible to isolate some of these layers in the case of the conference graph. In particular, we will show that:

- The communities behind conferences provide a good measure to relate conferences to each other.

- This measure consists of a thematic as well as a quality component – the major layers of our graph.

- The thematic layer can be identified by the mere analysis of publication titles.

- The quality layer can be partly isolated by subtracting the thematic component from the overall relationship.

As a result of the layer separation, it becomes possible to explore the conference graph under different points of view. We introduce a novel idea for conference rating based on the quality layer of the graph. Afterwards, we present a collaborative conference search website that demonstrates the advantages of having independent notions of the thematic scope and the quality of a conference. It offers different ways to search for conferences and can be fine-tuned to match the quality and deadline restrictions of a user. Thereby, it greatly assists researchers that are looking for a suitable place for submission.

We also use the proposed conference similarity measure to visualize the world of conferences, and to look into some trends in computer science. In particular, we try to draw a map of computer science. This map is thought to be interesting to look at, but surely contains errors and distortions. It does definitely not reflect the ultimate truth and should thus be used with care. Finally, we show how the relationships between different conferences have evolved over time. Again, this information is rather thought to be entertaining than exact. Consequently, the findings should be taken with a pinch of salt.

8.1 Related Work

The analysis of publication records has been an active field of research for a long time. Clearly, one of the most attractive goals for publication database mining is automated conference and journal rating. Garfield's pioneering work in 1972 [38], which describes the use of citation analysis for this purpose, initiated a long – and still ongoing – controversy. On one hand, many authors point out the wide variety of problems of the citation indexing approach [70, 76, 104]. On the other hand, citation analysis is presumably still the best method to automatically rate scientific conferences and journals. Other measures that are used to indicate a venue's relevance are the acceptance rate as well as time delays, such as turnaround time, end-to-end time, or reference age [111]. It seems that the community behind a conference has so far not been taken into account for automated rating. We believe that this criterion should not be neglected. In the following we thus provide an idea to fill this gap.

The rating of venues is not the only motivation for research on bibliometric data. Other insights have been gained from publication databases. One closely related aspect is the characterization of authors (rather than venues). Various measures, such as closeness [29, 82, 87, 72, 109], betweenness [29, 87, 72], or AuthorRank [72] have been evaluated in this context. Also, many studies analyze the evolution of different properties [29, 82, 109, 10, 63]. For us, the publications of Lee et al. [63] and Smeaton et al. [109] are of particular interest, as they study the topical changes within a single conference over the years. Thereby they show that the analysis of publication titles, keywords, and abstracts is sufficient to extract the thematic scope of a venue – a fact that we will take advantage of. Moreover, towards the end of this chapter, we also look into changes over time, on the one hand among different conferences, and on the other hand by predicting – in a tongue in cheek way – the perfect paper title for certain conferences.

Another perspective to looking at the thematic scope of venues is presented in [29]. By considering a common author of two venues an indicator for thematic similarity, a weighted graph is constructed that interrelates the most important conferences in the field of database research. We improve on this measure by incorporating some means of normalization and show that the thematic proximity is only one aspect contained in this weight.

Similarly as when people connect to friends, there are different reasons why Authors publish at a certain conference. In analogy to the layers (or social dimensions) identified by Watts et al. [124] in friendship networks (recall Chapter 3), these different reasons for choosing a place for publication lead to different layers in the conference graph. In the following, we identify quality and thematic scope as the two major layers of our conference graph, and propose a technique to separate the two components.

8.2 The Conference Graph

This section describes how the publication records of DBLP[1] can be used to generate a graph that interconnects scientific conferences. The graph construction bases on the social network behind these conferences. We basically assume, that the more common authors two conferences have, the more related they are. To avoid overestimating the similarity of massive events – they naturally have a large number of common authors – we improve on this idea by incorporating a normalization method: Consider two conferences, C_1 and C_2, that contain a total of s_1 and s_2 publications, respectively. Further, assume that there are k authors A_i $(i = 1, ..., k)$ that have published in both places and that author A_i has $p_{i,1}$ publications in conference C_1 and $p_{i,2}$ publications in conference C_2. We can now define the similarity $S(C_1, C_2)$ between C_1 and C_2 as follows:

[1] `http://dblp.uni-trier.de`

KDD		AAAI		ECAI	
ICDM	0.69	IJCAI	0.76	IJCAI	0.53
SDM	0.58	ATAL	0.37	KR	0.29
PKDD	0.45	ICML	0.33	ATAL	0.27
PAKDD	0.40	AGENTS	0.32	AAAI	0.26
ICML	0.37	AIPS	0.31	AI*IA	0.24
DMKD	0.37	ECAI	0.26	JELIA	0.22
CIKM	0.36	KR	0.25	ECSQARU	0.21
SIGMOD	0.36	UAI	0.25	CP	0.19
ICDE	0.35	CP	0.23	IEA/AIE	0.19
VLDB	0.33	FLAIRS	0.20	KI	0.19

Table 8.1: The 10 strongest links to the conferences KDD, AAAI and ECAI

$$S(C_1, C_2) = \sum_{i=1}^{k} \min\left(\frac{p_{i,1}}{s_1}, \frac{p_{i,2}}{s_2}\right)$$

The intuition behind this measure is as follows: p_i/s_j is the fraction of papers author i has contributed to conference j, and thus indicates how strongly the author is associated with the conference. The link weight between two conferences is given by summing up the contributions of all the common authors. Thereby, each author contributes the weight of the weaker of the two associations (thus the *min()* function).

Applying this similarity measure to all pairs of conferences results in the desired graph. The required information for this graph was extracted from the DBLP bibliographic repository. Any publications that appeared in a scientific conference between 1996 and 2006 have been taken into account. To reduce the amount of data, edges of extremely low weight that do not significantly contribute to the connectivity have been removed. To give a more concrete idea of the structure of this graph, Table 8.1 lists the 10 top edges for some sample conferences.

8.3 The Layers

Similarly as in the case of social networks, there exist different catalysts for edges. As in Chapter 3, we will refer to edges corresponding to different catalysts as the layers of our graph. We will have a closer look at two of these layers, namely the thematic and the quality layer, throughout this section.

Proximity in the conference graph is not purely defined by the thematic similarity of venues as a careful look at Table 8.1 reveals. *ECAI* is, for example, typically said to be thematically closer to *AAAI* than *ATAL*, *ICML*, or *AGENTS*, which appear earlier in the *AAAI* top-10 list. We conclude that

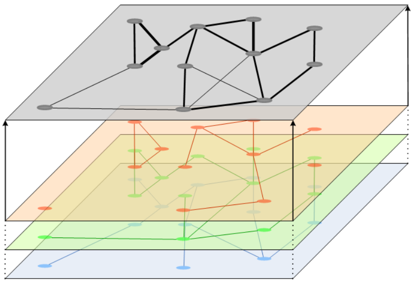


Figure 8.1: The total graph can be seen as the sum of its layers.

authors choose conferences not only because of the topic it covers. Other properties, such as quality, geographic location, or the community behind a venue also influence the author's decision. In fact, we believe that it is a weighted combination of all these factors that leads to a submission at a certain place. Exactly this combination is reflected by the conference graph presented in the previous section. That is, the graph consists of different layers, where each layer represents one of these factors. This idea is illustrated in Figure 8.1.

8.3.1 The Thematic Layer

Clearly, the thematic scope of a conference has a significant impact on its relationship to other venues. In the following, we present a technique that is based on publication title analysis and measures the thematic similarity of conferences. It thus allows to define the *thematic layer*, which is surely an ingredient of the social similarity measure, as a majority of authors mostly work in only one area and therefore submit papers to thematically similar venues.

For each conference, we have extracted all the titles from DBLP and applied the well-known *term frequency - inverse document frequency (TF-IDF)* method (see [100] for some theoretic background) to identify the most relevant keywords. The TF-IDF score for a document increases proportionally to the number of occurrences of the keyword in the document (TF). However, words that have a high overall frequency are penalized (IDF). In our context, a document corresponds to a venue and the words stem from publication titles. Consequently, a document consists of all titles in a venue and the complete corpus consists of all venues in DBLP.

Once a score has been applied to all the keywords that appear in the

KDD		AAAI		ECAI	
mining	0.051	learning	0.013	reasoning	0.011
data	0.013	planning	0.012	learning	0.010
discovery	0.013	robot	0.010	qualitative	0.009
clustering	0.013	reasoning	0.008	planning	0.008
association	0.010	knowledge	0.007	knowledge	0.008
sigkdd	0.010	search	0.007	logics	0.008
kdd	0.009	agent	0.006	logic	0.008
frequent	0.009	constraint	0.006	ecai	0.008
rules	0.009	ai	0.006	constraint	0.007
discovering	0.009	reinforcement	0.006	diagnosis	0.007

Table 8.2: The 10 best matching keywords to KDD, AAAI and ECAI together with their TF-IDF score.

KDD		AAAI		ECAI	
ICDM	26	IJCAI	37	IJCAI	29
PKDD	23	ECAI	27	AAAI	27
PAKDD	21	FLAIRS	22	ICTAI	22
SDM	20	ICTAI	21	KI	21
Dis. Science	20	AIPS	17	FLAIRS	20
DMKD	18	Can-AI	16	Can-AI	19
ADMA	17	IEA/AIE	16	IEA/AIE	18
ISMIS	17	PRICAI	15	PRICAI	18
IDA	15	Aus-AI	15	KR	16
IDEAL	15	KI	14	Aus-AI	16

Table 8.3: The 10 closest neighbors to KDD, AAAI and ECAI in the thematic layer, together with their thematic score.

conference's collection of titles, the scope of the conference can easily be estimated by looking at the most relevant terms. Table 8.2 shows some examples.

Using the keyword-lists seen before, we have implemented a simple algorithm that estimates the thematic relationship between two venues. It takes the top-50 keywords of each conference, and counts the number of keywords appearing in both lists, resulting in a score from 0 to 50 for each pair of conferences.[2]

Applying the thematic similarity function to each pair of venues results in a weighted undirected graph – the *thematic layer* of our graph. The corresponding neighborhood lists for our sample conferences are shown in Table 8.3.

[2]Surprisingly, this simple comparison function achieved slightly better results than the more commonly used cosine similarity approach.

AAAI Total		AAAI Thematic		ECAI Total		ECAI Thematic	
IJCAI	1.10	IJCAI	1.10	IJCAI	1.10	IJCAI	1.10
ATAL	*1.51*	ECAI	0.69	KR	1.76	AAAI	1.49
ICML	*2.12*	FLAIRS	N/A	*ATAL*	*1.51*	ICTAI	0.25
AGENTS	*1.00*	ICTAI	0.25	AAAI	1.49	KI	0.41
AIPS	1.53	AIPS	1.53	*AI*IA*	*0.26*	FLAIRS	N/A
ECAI	0.69	Can-AI	0.26	*JELIA*	*0.72*	Can-AI	0.26
KR	*1.76*	IEA/AIE	0.09	*ECSQUARU*	*0.38*	IEA/AIE	0.09
UAI	*N/A*	PRICAI	0.19	*CP*	*1.04*	PRICAI	0.19
CP	*1.04*	Aus-AI	0.16	IEA/AIE	0.09	KR	1.76
FLAIRS	N/A	KI	0.41	KI	0.41	Aus-AI	0.16

Table 8.4: The 10 closest neighbors to AAAI (left) and ECAI (right) in the total graph and the thematic layer, together with the Citeseer impact value. Note that for AAAI, conferences in the total graph neighborhood that are not present in the thematic layer list (*italic*) all have relatively high impact value. The impact value of such conferences in the neighborhood of ECAI is considerably lower.

8.3.2 The Quality Layer: Filtering by Subtraction

We have already quickly sketched different approaches to conference rating and stated some of the difficulties (recall Section 8.1). For computer sciences, the Citeseer Impact List[3] tries to estimate the impact of venues based on citation analysis. Further, many researchers maintain hand-made lists that distinguish between tier-1, tier-2, and tier-3 conferences. Even though hand-made lists suffer from a subjective bias and citation analysis from other weaknesses (recall Section 8.1), tier-1 conferences typically have a high impact and, contrariwise, tier-3 conferences get low scores in the Citeseer list. We will refer to similarly classified conferences as conferences of similar quality.

Comparing the neighborhood tables for the total graph (Table 8.1) and the thematic layer (Table 8.3) shows that the total graph is *not* purely defined by the thematic correlation of conferences. Looking at the total graph, an interesting observation is that conferences often considered to be of high quality (such as *KDD* and *AAAI*) tend to have other high quality conferences in their proximity. In contrast, the number of lower-tier conferences in the proximity of *ECAI*, which is mostly classified as tier-2, is significantly higher. This observation is illustrated in Table 8.4 that uses the impact value of the Citeseer Impact List to classify the conferences.

We conclude that a single author tends to publish not only in venues of

[3] `http://citeseer.ist.psu.edu/impact.html`

similar topic, but also in venues of similar quality. As a result, our graph contains a second major layer – the *quality layer*.

The observation that thematically weaker related nodes in a conference's proximity tend to be closer in quality suggests that the quality layer can be extracted using the information about the total graph and the thematic layer. In the following we will introduce a *layer subtraction approach* to demonstrate that such a layer separation can indeed be achieved. The approach bases on the (simplifying) assumption that the total graph is a linear combination of the single layers. As a result of the observations in the previous section we assume that the major layers of the conference graph are the thematic layer t and the quality layer q. This also matches our experience when selecting a conference: We make sure the publication matches the call for papers and we try to submit at a conference of reasonable quality. Other factors, such as geographic location, play a minor role in the decision. These factors (including noise) are thus subsumed into a remainder layer r. Consequently the total edge weight S becomes to $S = \alpha_1 \cdot t + \alpha_2 \cdot q + \alpha_3 \cdot r$, for some weights α_i, with $\alpha_1, \alpha_2 \gg \alpha_3$. Neglecting α_3 and setting $\alpha_2 = 1$ (α_2 can be chosen arbitrarily as it only results in a scaling of q) allows to extract the quality layer q as

$$q \approx S - \alpha_1 \cdot t,$$

Note that the validity of the linear combination assumption greatly depends on the characteristics of the weight functions in the different layers. Fernandez et al. [32] present the idea of score distribution normalization for aggregation purposes. They suggest to shape the histograms of the independent score functions to match the "ideal" distribution prior to merging them by linear combination. For simplicity we assume a uniform weight distribution for both, the total as well as the thematic scores.

Observe that the subtraction approach generally allows to extract one out of L layers of a graph, if the remaining $L - 1$ layers are known. As discussed in Chapter 3, it seems that such a layered structure can often be observed in natural graphs. Moreover, recommendation systems (such as integrated in the Amazon online store) often build on similar co-occurrence structures as our graph and are thus likely to exhibit similar properties. Hence, we believe that the layer-subtraction approach can prove valuable as a preprocessing step in various settings.

8.3.3 Interpolation Based Conference Rating

The proximity of a conference in the quality layer is supposed to contain mostly conferences of similar quality. This observation immediately leads to the idea of conference rating by interpolation: Provided some initial ratings are known, the tier of a conference can be estimated by looking at its proximity in the quality layer. Initial ratings can be retrieved from manually created lists (we use the one found at `www.ntu.edu.sg/home/assourav/crank.htm` and refer to it as *CS Rating List*) as well as from Citeseer's impact list. We have further introduced the *Citeseer Tier List*, which assigns a tier (1, 2, or 3) to each conference in the Citeseer Impact List. The borders between tiers have been chosen such that the number of incorrectly rated conferences with respect to the CS Rating List becomes minimal. The best that can be achieved is an error rate of 38.8%, which indicates how difficult the task of conference rating is.

We have then defined a heuristic to rate a conference C_0 as follows:

1. For all conferences in the CS Rating List or the Citeseer Tier List, set the initial rating to the value found in the lists. In case of conflicts, the CS Rating List is treated with priority. For any conference not in the lists, set the initial rating to *unrated*.

2. Overwrite the initial rating of C_0 with *unrated*. This step avoids that the rating function is biased towards the initial value.

3. Take the 30 shortest edges e_i adjacent to C_0 in the total graph, together with their values S_i and t_i. For all these edges, calculate $q_i = S_i - \alpha_1 \cdot t_i$ (for some value of α_1) and sort by q_i in ascending order. We will call the resulting list the *filtered neighborhood list* of C_0: $N_f(C_0)$.

4. For the first 5 entries C_j $(j = 1..5)$ in $N_f(C_0)$, calculate $N_f(C_j)$.

5. Return the median of all the rated conferences found within the first 5 entries in all the lists $N_f(C_j)$ $(j = 0..5)$ as the rating of C_0.

Note that this conference rating method is in some sense natural. Many people would judge a venue based on people participating in it (or leading it). This information is implicitly contained in the total graph which forms the basis of the rating heuristic.

The quality of the heuristic can be estimated by comparing the calculated ratings to those found in the CS Rating List (which is presumably the most accurate list we dispose of). The optimal value of α_1 was scanned for by exhaustive search over some reasonable interval. This is illustrated in Figure 8.2 which plots the error rate of the rating function with respect to the CS Rating List for different values of α_1. The figure clearly shows that the subtraction approach reduces the number of incorrect ratings and suggests that the optimal value of α_1 is somewhere between 0.5 and 1.

Arguing with error rates beyond 40% might at the first glance seem suspicious. However, the fact that approximately 75% of the input values (namely

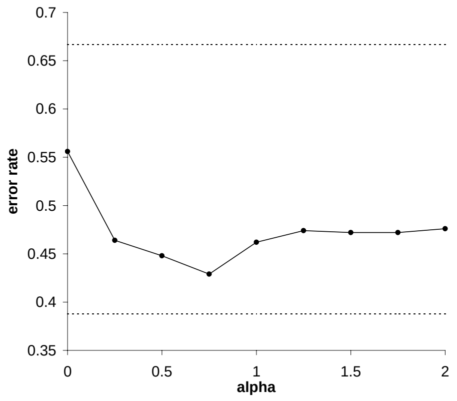


Figure 8.2: The fraction of incorrectly rated conferences using our rating function versus the value of α_1. The dotted lines indicate the error rates for a random guess (0.667) and for the Citeseer Tier List (0.388), which is in about the best we can expect to reach as most of the initial rating values stem from this list.

those that originate from the Citeseer Tier List) exhibit an error rate of approximately 40% themselves relativizes the high error rate produced by our algorithm.

Ignoring all the conferences rated as tier-2 either by the algorithm or the CS Rating List shows that the errors are not random. Dividing the number of conferences rated as tier-1 instead of tier-3 (and vice versa) by the number of conferences the algorithm rates as tier-1 or tier-3 results in an error rate of around 6.4% without thematic filtering and of 2.6% for the optimal value of α_1.

These low error percentage values show three things:

1. The total edge weight is clearly influenced by the quality of conferences. This supports the assumption that the thematic and the quality layer are the two main layers of the graph.

2. The success of extracting the quality layer by subtraction of the thematic layer is confirmed.

3. Most of the around 43% of errors are minor errors. That is, they are wrong by only one tier. Severe errors are rare, they make up less than 3%.

The rating heuristic was developed for two reasons: To provide a complete rating list for the conference search application presented next, and to demonstrate the effect of subtraction filtering. It is thought as a proof-of-concept algorithm that neither has a strong mathematical foundation nor provides any guarantees on the results.

8.4 Confsearch

In this section we will show that the previously discussed conference graph and its separation into different layers can directly be applied for conference search. For this purpose we have developed a website (`http://www.confsearch.org`) that is able to suggest conferences together with their most important attributes. The application offers four different search types:

- *Keyword Search*: Search by keywords provided.
- *Related Conference Search*: Explore the proximity of a given conference in the conference graph and return the closest neighbors.
- *Author Search*: Search for the places a given author publishes most often.
- *General Search*: A weighted combination of the above search methods.

For all search types the application allows to sort the results by deadline, a criterion that has a considerable impact when deciding for one or the other venue. Motivated by the success of Wikipedia like services, we follow a collaborative approach to gather conference deadlines as well as locations and website URLs. The important dates (submission deadline, notification, camera ready version, and conference dates) are presented in a Gantt chart thereby providing a good overview over the conferences' time schedules. Our application can be seen as an improvement on the many lists with conference deadlines found in the Internet today: We basically cover the whole area of computer science and augment the typically static lists with sophisticated search options. Moreover, the information is presented in a visually appealing manner.

Two of the search modes require some additional explanations. The *keyword search* bases on a score s_{ij} for each keyword-conference pair (where only keywords appearing in the query are considered), which is a slightly modified variant of the TF-IDF value presented in Section 8.3.1. Next, the scores s_{ij} of the conference-keyword pairs are combined to a single value S_i^* per conference C_i using the *p-norm* method introduced by Salton et al. [101].

$\beta_q = 0.0$	$\beta_q = 0.5$	$\beta_q = 1.0$
PKDD	KDD	KDD
KDD	ICDE	ICDE
INFOVIS	PKDD	ICDM
ICDM	ICDM	VLDB
Web Intelligence	Web Intelligence	Web Intelligence
PAKDD	INFOVIS	PKDD
ICDE	VLDB	DMKD
ICDM	DMKD	SDM
JSAI Workshops	SDM	INFOVIS
DaWaK	PAKDD	DASFAA

Table 8.5: The results for the search query “social graphs data mining” for different quality weights (controlled by the parameter β_q).

The final score S_i results from the quality adjustment of S_i^* controlled by a user-settable parameter β_q: $S_i = S_i^* \cdot f(Q)^{\beta_q}$. The function f(Q) is defined on a per query basis to account for the different score distributions for different queries. The quality part Q is estimated using the heuristic presented in Section 8.3.3. Table 8.5 presents a keyword search example and the effect of quality filtering.

The *related conference search* operates directly on the conference graph. We simply return the closest nodes around a conference in terms of path length. Again, a user settable parameter allows to control whether the thematic or the quality aspect should be emphasized. A visualization of the *AAAI* neighborhood in the thematic and the quality layer can be found in Figure 8.3. The increased amount of high quality nodes (dark) in the *AAAI*'s “qualitative proximity” indicates that *AAAI* itself is also likely to be of high quality. The search option on one hand allows to browse the conference graph and on the other hand might prove extremely helpful if an author looks for alternative places to submit, after a reject, for example, or because a deadline does not fit.

In the following section, we will not only visualize the direct neighborhood of a conference, but we will try to capture the global picture, by drawing a map of the world of conferences.

8.5 Computer Science Cartography

Maps have always been an essential instrument in exploring space. It was, after all, the Ptolemy world map – and its mistakes – that triggered Columbus' monumental voyage and the discovery of America. Reason enough to draw a map of “our world”, the world of computer science. Being more risk-averse

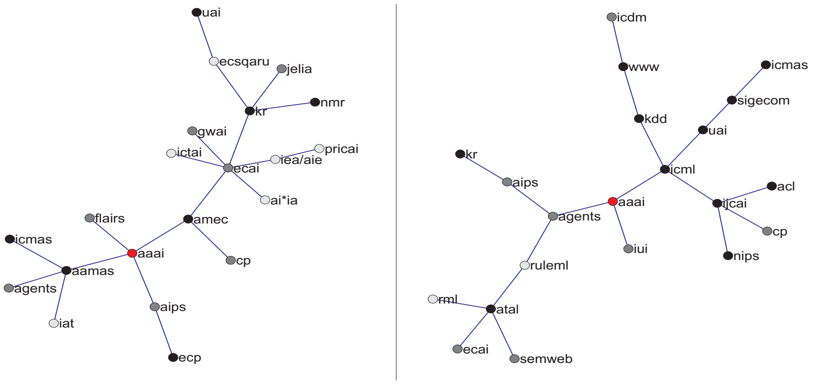


Figure 8.3: The minimum spanning tree around $AAAI$ in the thematic layer (left), and the quality layer (right). Darker nodes refer to higher tier venues.

than Galileo et al.[4] we would like to stress that the following investigations should be taken with a grain of salt.

In Section 8.2 we have seen how our conference similarity measure can be used to construct a conference graph. In the following, we will use this graph to draw a map of computer science, that visualizes the relationships among the different fields in computer science.

Luckily, the "cartography of graphs", better known as graph embedding, is a well explored topic. Simply speaking, the goal of a graph embedding algorithm is to assign Euclidean coordinates to all vertices of a graph, such that the resulting positions well reconstruct the graph distances between all pairs of nodes (also multi-hop).[5] Similarly as it is impossible to undistortedly draw the globe in 2 dimensions, it is in general also not possible to exactly represent a graph metric in 2 dimensions.

Out of the rich choice of embedding algorithms we have opted to apply the widely used multi-dimensional scaling method (MDS) to create our map. More precisely, we apply classical MDS that tries to approximately preserve all the pairwise distances. That is, we seek to assign coordinates to the nodes (conferences, in our case) of a graph such that pairwise distances in the original graph metric (i.e. shortest paths) and the resulting Euclidean metric are approximately the same.

[4]Not to mention Giordano Bruno!

[5]More generally, graph embedding is the process of mapping a graph into any other space (not necessarily Euclidean), thereby pursuing some (not necessarily distance related) design goals.

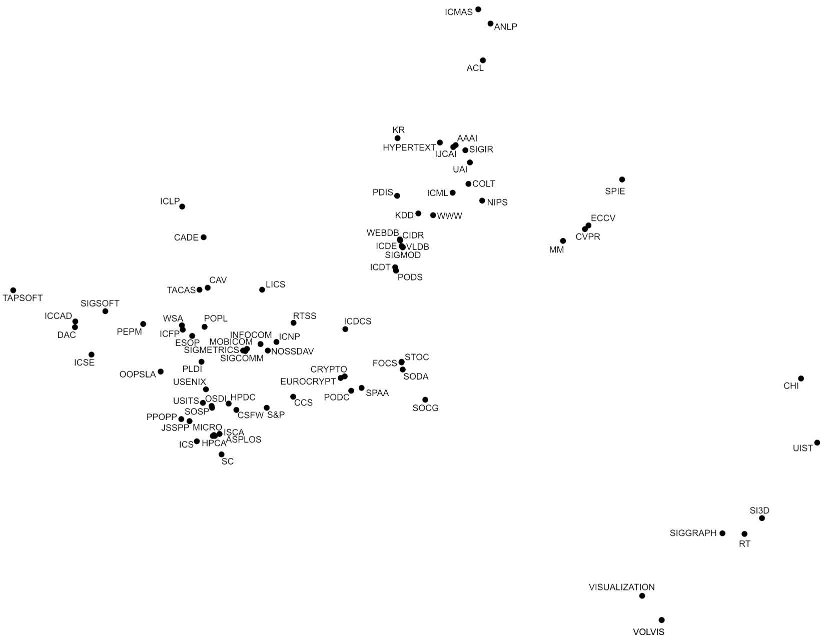


Figure 8.4: Our map of computer science: The map was constructed by embedding the conference graph into a 2-dimensional Euclidean space. Only top-tier conferences (according to Libra) are shown. Note that the map only represents pairwise distances, there is no notion of orientation, i.e. the axes can be chosen arbitrarily.

It is important to note that there is no notion of orientation for such an embedding. It only represents pairwise distances, and would be equally valid after applying any congruence transformation.

Figure 8.4 shows the resulting map of computer science conferences. To keep the map readable we have restricted the vertex set to only include top-tier conferences. The required conference classification – with which we disagree to a certain extent – was taken from *Libra*[6].

Our map gives room for quips and questions. It can, first of all, be observed that like-minded conferences tend to gather together, they build families and tribes. It is thus possible to investigate the interconnections

[6]`http://libra.msra.cn/`; For each discipline the site used to list the major conferences grouped by tier in 2008.

between entire "scientific cultures" rather than single conferences. Software engineers and design automation specialists, for example, live in the west of this world.[7] Their eastern neighbors are mainly into computer networks, and further south hardware architecture and operating systems experts are located.

In the very north, people mainly study natural languages and artificial intelligence. Also, they retrieve information from and mine the data provided by their direct southern neighbors (databases and the world wide web). The south-eastern population, finally, lives quite independently, and is mostly interested in user interfaces, graphics and visualization. It is interesting that these disciplines almost "drop" from the disk, and that we experience these large distances in the south-eastern part of our map. Are these gaps merely an artifact of our sloppy model, making our drawing as inaccurate as the disk-shaped Babylonian world map, or does this emptiness call for attention by the computer science community?!

We observe that the center of the graph is dominated by theory, and – slightly less significantly – also cryptography, distributed computing, networks, and databases. Interestingly, all these areas can be seen as service suppliers for other disciplines: Data can hardly be mined without databases, software design requires a great amount of (algorithmic) theory, and networks as well as cryptography and distributed systems play a crucial role in many real world systems. Hence, looking at our map, a theoretician could, with a healthy dose of self-esteem, derive that he (or at least his field of research) is in the center of computer science.[8]

Our theoretician would surely like to get a more detailed view on his self-declared center of computer science. Even though we question the center predicate, we do him – and hopefully also all the theory-focused readers – a favor and zoom into this section. We have inserted the lower-tier conferences (again, according to Libra, and again, we do not fully agree) into the drawing by placing them into the weighted center of their closest top-tier neighbors (while keeping the layout of these top-tier conferences fixed). Figure 8.5 illustrates the theory close-up of our map.

In an attempt to interpret the figure, one might notice a slight separation of west and east. The west-side of the map seems to be mostly populated by the species of "practical theoreticians", while the east-side is rather inhabited by "pure theory". Interestingly, the cryptographers squeeze themselves into

[7] Again, note that there are no axes defined, we just orient ourselves as the picture is aligned here.

[8] Clearly, the map leaves room for discussion. A closer look reveals many surprises, e.g. the location of design automation conferences. Also, other (equally reasonable) centers, such as databases or networking could doubtlessly be identified. More fundamentally one might wonder whether there at all is a center of computer science; maybe we rather live in a "centerless" world, much like modern physics sees the universe.

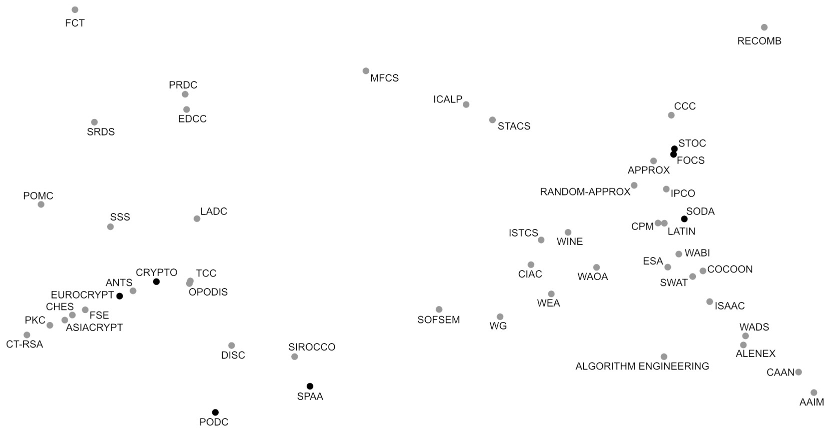


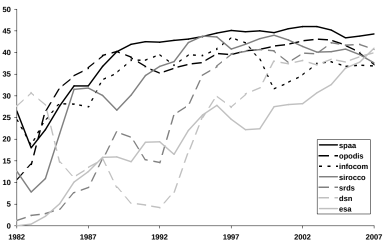
Figure 8.5: Close-up of the map around the theory conferences. Tier-1 conferences (according to Libra) are marked black, any other conferences gray.

the territory of distributed computing. Clearly the two share common roots, but still the proximity is remarkable. Comparing the distances in the map to the distances in the original graph reveals that there is quite some discrepancy in this specific case. We speculate that distributed computing and cryptography get intermixed because they both lie between (pure) theory and systems/networking. Possibly, a 2-dimensional map can not accurately represent the ultimate truth. As detailed views usually provide deeper insights into a problem, the insight of this close-up is perhaps that our map should be treated with care.

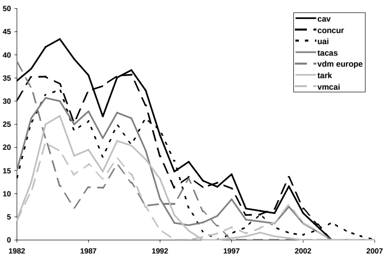
8.6 Migration in Computer Science

As the history of mankind, the world of computer science is not static. Communities emerge, disappear, and migrate. In the following we want to have an eye on the movements in the proximity of PODC, as well as some major theory conferences, namely STOC, FOCS and SODA (which we in the following will treat as one).

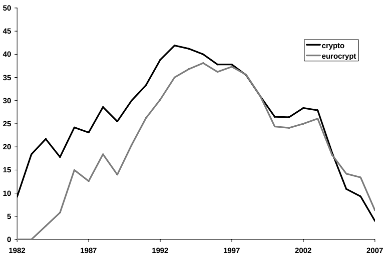
A conference is mainly defined by its participating authors. We thus assume that looking at the changes in authorship is a handy method to capture the changes of a conference over time. Consequently, we have applied the idea of our "social similarity measure" from Section 8.2 on a per year basis: For a particular year and conference we have examined where else the



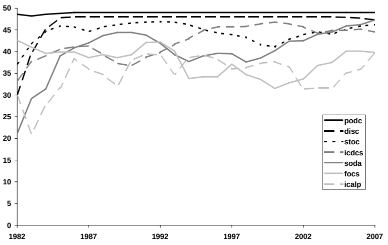
(a) Conferences getting closer.



(b) Conferences losing contact.



(c) Temporary close.



(d) Evergreens.

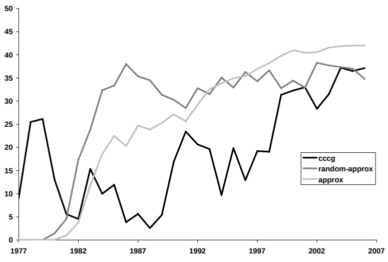
Figure 8.6: Related conference trends for PODC.

authors would typically publish (at all times). This gives, for this particular conference, an insight in what other conferences have been particularly close at a given point in time.

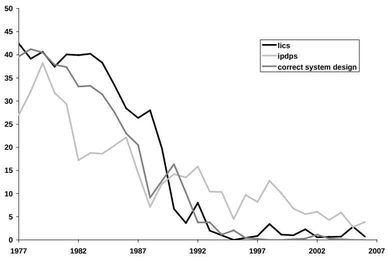
The described "time dependent conference similarity measure" allows to plot the development of a conference's proximity over time. Figures 8.6 and 8.7 show such plots for PODC and STOC/FOCS/SODA. An interesting observation is, maybe, the temporal closeness of cryptography to both, PODC and the theory conferences. Most likely, this does not mean that cryptography is in danger of extinction, but rather protocols an emancipation process; cryptographers have grown strong enough to form their own, independent community and thus drift away from other conferences.

8.7 Predicting the Perfect Paper Title

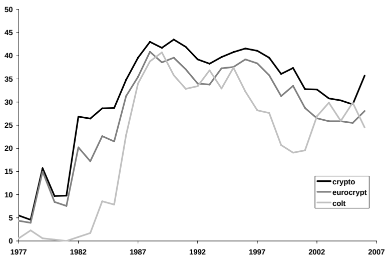
After this brief excursion into the evolution of conference relationships, we want to come back to a more global view. How did computer science evolve over time? We believe that terminology is an meaningful witness for scientific history. We thus analyze the change of central keywords in computer science.



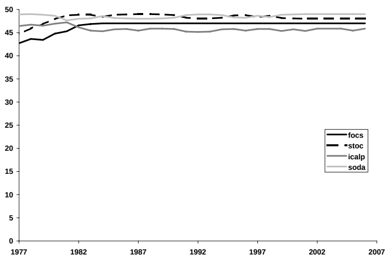
(a) Conferences getting closer.



(b) Conferences losing contact.



(c) Temporary close.



(d) Evergreens.

Figure 8.7: Related conference trends for STOC/FOCS/SODA.

Moreover, we will, in a tongue-in-cheek way, suggest the perfect titles for upcoming editions of PODC as well as STOC/FOCS/SODA.

As usual when it comes to predicting future trends we rely on historic data. For this purpose, history has been divided into 6 time-slots, from 1977 to 2006 in 5-years chunks. Moreover, for each conference in question we have parsed all the titles appearing in its proceedings to extract the single words. Counting the number of occurrences for each word and time-slot allows to establish a "per-time-slot" ranking which can then be mined to extract trends. We have restricted the search space to keywords that made it into the top-50 in at least one time-slot. The actual trend analysis (i.e. selection of the keywords that best characterize the movements) was then mainly carried out manually. In the following, we propose paper titles from these keywords. These titles are purely made up to contain the relevant keywords and do not have any deeper meaning.

Our analysis shows that *wireless mobile agents based on neural learning that quickly adapt to image and video web services* are fashionable.[9] On the

[9]In other words, the keywords web, agent, wireless, neural, mobile, video, learning,

other hand, we are no longer interested in *computer experts specifying the semantics of parallel relational databases for VLSI in the Prolog programming language.*[10] Also, *knowledge about object oriented simulation logic* does not seem to be required any longer, even though the topic was a hot in the beginning of the 90ties.

If you do not want to risk being labeled antediluvian after submitting to a major theory conference, better name your paper, say:

> *"Online Quantumn Algorithms to Approximate Directed Location Codes"* or *"On the Hardness of Scheduling using Random Sampling"*

Moreover, avoid titles like

> *"Relational Logic to Separate Automata Isomorphism Classes"* and *"Probabilistic Parallel Programs for VLSI Design"*

as these will immediately out you as being stuck in the 80ties. Similarly, the next year's blockbuster of PODC is rather going to be called

> *"Failure Detectors and Scalable Dynamic Quorum: A Free Mobile Ad-Hoc Game with Selfish Peers"*

than, for example

> *"Recover from Committed Ring-Deadlocks using Message Passing Communication, Temporal Knowledge and Parallel Computation Processes"*

8.8 Conclusion

What is computer science all about? A controversial question. Some might claim that computer science is all about building computing machines – possibly only theoretically. Others think it is all about the art of programming these machines, or about describing languages to talk to them. Yet other people associate computer science with algorithms, and believe P versus NP is at the heart of computer science. Finally, the tremendous impact of the Internet, or the huge data collections in service today might speak in favor of networking or database experts, respectively.

Throughout this chapter we have investigated the relationships among different computer science disciplines by looking at the communities behind conferences. In particular, we have defined a similarity measure to relate

image, adaptive, services are becoming increasingly popular.

[10]Again, we witness that the keywords computer, expert, etc. have seen their peak.

conferences, and we have used this measure to construct a conference graph. Using this graph we have provided evidence for the layered structure of real-world networks as discussed in Chapter 3. We have seen that the conference graph consists of two major layers – the thematic layer and the quality layer. Moreover, we have proposed the subtraction approach for layer segregation, which provides an attractive preprocessing step when mining graphs. In our setting it was used to accentuate the different aspects of conference relations. The separation approach has directly been applied in a novel rating method for scientific conferences.

The conference graph also builds the basis of confsearch, a novel conference search application, designed to assist researchers in finding a suitable place for submission. Moreover, we have used the proposed conference similarity measure to draw a conference map that visualizes the relationships among different scientific fields, and to look into changes in the relationships among conferences over time. These last applications of the conference similarity measure provide some interesting insights into the world of computer science. However, we want to stress that they are not supposed to reflect the ultimate truth. Rather, they might serve as a starting point for discussions about our work and about computer science in general. We have looked into the evolution of scientific disciplines and observed that the same evidence can be interpreted as both, the birth as well as the funeral of a research area. Without doubt the future will teach our evaluations a lesson, ultimately revealing in which direction computer science evolves. After all, research is not about how many papers we write, or at which conferences they are published, but rather, what the best contributions are.

Chapter 9

Mapping Music Similarity

Over the past years we have observed a tremendous change in the way people interact with music. This process has been ignited by several technological advances, in particular, the availability of broadband Internet, the world wide web, affordable mass storage, and compact media formats, such as `mp3`. All this enabled the digital music revolution that started more than decade ago.

Many music lovers have now accumulated collections of music that have reached sizes that make it hard to maintain an overview of the data by just browsing hierarchies of folders and searching by song title or album. Search methods based on song *similarity* offer an alternative that allows users to abstract from manually assigned metadata, such as, frequently imprecise or incorrect, genre information. In a context where music collections grow and change rapidly, the similarity based organization has also the advantage of providing easy navigation and retrieval of new items, even without knowing the new songs by name. Moreover, it allows personal collections to be seen not just as isolated lists, but positioned in the global context of the "world of music", i.e., to relate personal music collections to larger collections and newly-released music. This opens possibilities, such as sophisticated recommendations, context-aware retrieval, and discovery of new genres and tendencies.

This chapter introduces the concept of a Euclidean map as a basic data structure for music exploration and retrieval. In contrast to existing approaches that go into similar directions, our primary goal is not visualization. Rather, we aim at adequately reflecting, and compactly representing similarity between songs and genres.

In the following we propose two different methods to construct such a map from the information contained in the music community site *last.fm*. The first approach follows the ideas we have already seen to construct the conference map. In particular, we make use of the following 3-step process: First, we estimate song-to-song similarities using collaborative filtering techniques. Second, we construct a graph from these pairwise similarity values. Third, we map the graph into Euclidean space while approximately preserving distances. For the last step, we introduce a novel technique (called *iterative embedding*), which improves on existing algorithms.

The second map construction approach does not solely rely on listening behavior to derive music similarity, but in addition exploits information from social tagging. In particular, we will describe a method that combines the two signals before they are fed into a statistical framework called Probabilistic Latent Semantic Analysis (PLSA). PLSA basically performs dimensionality reduction on co-occurrence data by identifying a (small) set of hidden variables that well explains the observed co-occurrence values.

The resulting maps comprise of more than 400K and more than 1M songs, respectively – an application foundation, which to the best of our knowledge is more comprehensive than other existing approaches. For these maps, we provide an analysis of how well they capture music similarity. Thereby we show, amongst others, that neither two nor three, but rather ten or more dimensions are required to appropriately map the world of music. In Chapter 10 we will discuss several mobile user interfaces that demonstrate the usefulness of our maps in realistic settings.

9.1 Related Work

Deriving a notion of music similarity is the subject of a variety of research activities. In the following we will try to give a rough (but non-exhaustive) overview of approaches that have been proposed.

These approaches can be classified along different criteria. Berenzweig et al. [13] distinguish between acoustic and subjective approaches. Under acoustic approaches they understand all kind of similarity measures that are based solely on audio-signal analysis. This kind of similarity information does not involve any human judgment and can thus be seen as an objective measure. Under subjective measures, on the other hand, they understand any kind of techniques that involve human interaction, such as the analysis of expert assigned metadata, collaborative filtering based techniques, questionnaires, and so on.

A significant part of research has been devoted to objective measures based on audio-analysis. Examples are the work of Logan and Salomon [74], Aucouturier and Pachet [4], Foote [34], Pampalk et al. [93], Tzanetakis [119], and Tsunoo et al. [116]. A good overview and discussion of such techniques is given in [17].

Audio-signal based music similarity measures exhibit some appealing properties. Clearly, they are not influenced by any subjective bias. Moreover, the extracted features often define some sort of a space, i.e. the extracted features can typically be interpreted as a vector. The basic assumption is then that similar feature vectors imply similar music, i.e. that similar songs cluster in this space. The nature of these spaces provides a powerful basis to construct novel interfaces to access music. Visualization of collections thereby plays an important role (see e.g. [58] and [83]), but also other interfaces, for example for intelligent playlist generation, have been proposed. A more thorough discussion of relevant work in this area is given in Chapter 10.

Audio-feature based techniques do, however, also exhibit some major disadvantages. Although objectivity might be advantageous some scenarios, the lack of subjective information proves to be a problem in most real-world settings. After all, music is typically targeted at people, and the perception of music is inherently subjective. To bridge the gap between abstract features and the end user, many approaches try to find a mapping between audio features and some widely used genre taxonomies (see e.g. [119] and [116]). A comparison between different audio based measures and their performance with respect to perceived similarity is given in [92]. Clearly, a certain success can be achieved when using audio-features for genre classification. However, Aucouturier and Pachet [7] conjecture that the currently used techniques and their variants have reached a plateau with respect to accuracy that can not easily be overcome. Moreover, they show that errors produced by state-of-the-art methods are often severe, i.e. misclassified songs can be completely different from their neighbors in terms of perceived similarity. The conclusions of Casey et al. [17] go into a similar direction. The authors state other studies that confirm the mentioned performance ceiling. Moreover, they identify scalability issues for audio-feature based methods when it comes to cover millions of tracks, as nowadays required in many applications. Slaney and White [108], finally, compare audio based methods to collaborative filtering based similarity measures. In their experiments, the investigated collaborative filtering based approach clearly outperforms the content based alternative.

Subjective methods encompass a wide variety of techniques. Sources of information range from expert assigned metadata, over (web-)text documents describing music and questionnaires, to games and usage data, such as applied in collaborative filtering. We will discuss collaborative filtering based techniques in more detail later. Questionnaires are mainly used as a ground truth to compare other methods, such as in [13]. Examples for the other fields are: [97] (metadata), [126] (web-text), and [28] (game). Sometimes, different techniques can also be combined. There are, for example, games that aim at collecting metadata (e.g. [62] and [77]) that can in turn be used to define music similarity. An approach that makes use of metadata to define similarity is presented by Levy and Sandler [66]. Their approach uses social tags, rather than expert assigned information, as metadata. Moreover, the authors stress the advantages of social tags over web-mined information with respect to noise and scalability. A thorough comparison of different sources to collect tags is provided by [117]. The results show that social tags suffer from a so called popularity bias, i.e. they offer good quality for famous songs, but are of limited use to describe less known items.

In an attempt to define a notion of ground truth, Ellis et al. [28] as well as Berenzweig et al. [13] compare different methods to derive music similarity. The studies include acoustic methods, expert opinion (similar artist lists from the *Allmusic Guide*[1]), co-occurrence information (from playlists and user collections), web-text information, and web-based games. The results of Berenzweig et al. indicate that co-occurrence information provides the best results among the subjective techniques. The investigated method to derive music similarity from co-occurrence information closely follows the concept of item-to-item collaborative filtering [71]. It is thus also closely related to the conference similarity measure discussed in Chapter 8. Another approach that follows a similar principle is the analysis of playlists of professional DJs, as presented in [99]. If two songs often appear next to each other in such playlists, they are likely to be similar. These algorithms can be seen as a subset of the more general collaborative filtering techniques that are widely used in recommender systems. In fact, many of today's commercial music services, such as last.fm, iTunes, Amazon, and iLike rely on co-occurrence information and collaborative filtering to provide music recommendations.

The discussed studies indicate that item-to-item collaborative filtering provides a better measure for (perceived) music similarity, than objective approaches. However, the resulting information cannot be used in an as versatile fashion as the feature-spaces produced by audio-analysis. In particular, item-to-item collaborative filtering only produces a weighted list of neighbors that co-occur at least once with the item in question. Hence, the method does not define a global space. Such a space, however, proves extremely helpful when designing smart interfaces to access music. This is, presumably, the reason, why most approaches that have recently been proposed in this context are based on audio-feature spaces (see also Chapter 10).

With the work we present in this chapter, we try to bridge this gap. In particular, we propose two methods to create music similarity spaces based on techniques from usage-data analysis and collaborative filtering. The first method closely follows the idea of item-to-item collaborative filtering, based on the users' listening behavior. A similar approach was followed by Gleich et al. [20] that presents a visualization of artist similarities. However, the goal of their work was an appealing drawing rather than the accurate positioning.

[1] `http://www.allmusic.com`

Moreover, it covers a much smaller universe of music than our maps do (less that 10K as opposed to more than 100K artists). That is, their work is not designed to build novel music retrieval interfaces and thus not appropriate for this purpose.

Our second map construction method makes use of two signals to define similarity: The users' listening behavior, and the social tags generated by a large listening community. The approach thereby adds to the work presented in [66], which only considers social tags. By incorporating the listening behavior we can reach a better accuracy and to a great extent overcome the popularity bias problem reported by Turnbull et al. [117]. At the same time, the resulting space keeps the advantages of the tags and their intuitively understandable meanings.

In Chapter 10 we will show that the proposed spaces can in fact be used in a similar way as audio-feature spaces. In particular, we propose interfaces for smart playlist generation and the visualization of music collections.

9.2 Method 1: Map Creation using Graph Embedding

The basic idea of our first map construction method is as follows: We begin by calculating pairwise similarities of (typically) relatively similar songs. These similarities are then seen as weighted edges in a graph. As a result, we can use the shortest-path metric on this graph to get a notion of similarity among all pairs of nodes. In a final step, the shortest path distance information is mapped into Euclidean space using a graph embedding algorithm, thereby defining coordinates for each song, and thus a map of music. In the following, this procedure is discussed in detail.

To obtain similarity values between songs we rely on collaborative filtering techniques. Similarly to how Amazon uses the fact that two items are related because they have been purchased by the same person, we assume that two songs are related if they are frequently listened to by the same user. We have gathered the required usage information from *last.fm*, which is a music-community site that counts over 20 million users, and records each user's listening patterns. In particular, for each user, the 50 most frequently listened songs over a half year period can be queried. We will refer to these lists as *top-50 lists.* The following discussion is based on 290K of these lists that contain a total of more than 1.5 million distinct songs. We assume that songs that co-appear in such a list exhibit some degree of relatedness, in the same way as items that are bought by the same person do. Observe that exceptions to this rule are typically random. That is, it is about equally likely that a person listens to Bach and U2, as it is that a person listens to Bach and Eminem. In the co-occurrence analysis, such "errors" therefore result in random noise, which does not significantly affect the outcome.

Simply counting the number of co-occurrences of two songs to calculate the pairwise similarities would overestimate the similarity of popular songs, as they clearly have a higher probability of appearing in the same top-50 list (due to their high number of individual occurrences). To overcome this problem, some sort of normalization is required. Several coefficients have been proposed that address this issue [78]. We have compared the performance of *cosine*, *dice*, *jaccard* and *overlap* coefficients, and found that the cosine coefficient performs best in our setting. It is defined as

$$\textit{cosine coefficient}\text{: } c = \frac{n_{i,j}}{\sqrt{n_i} \cdot \sqrt{n_j}},$$

where n_i denotes the number of occurrences of song i, and $n_{i,j}$ is the number of co-occurrences of songs i and j.

Applying the inverse of the cosine measure ($1/c$) to all pairs of songs results in a weighted graph that contains an edge between any two songs that appear together in at least one top-50 list. Thereby, similar songs are connected by short edges. To get rid of random effects any edges originating from a co-occurrence value of less than 2 have been removed. This step also eliminates any songs that occurred only once. Even after this step, the graph is extremely big, making it difficult to handle. Therefore, it has been made sparse using an edge weight threshold, which was defined such that the overall connectivity (i.e. the size of the largest connected component) was only marginally affected. The result is a graph G which contains $n = 430\text{K}$ nodes and $m = 6.3\text{M}$ edges. Using this graph, the similarity between songs is approximately given by the shortest path between them.

Due to the large size of our graph even simple operations, such as shortest-path calculations, are computationally expensive. In order to efficiently use such a large graph in (possibly mobile or distributed) applications, we go one step further and create the "map of music", which is an *embedding* of the graph into a Euclidean space. An *embedding* is the assignment of coordinates to each node of the graph. In our case, the goal is to approximately preserve all pairwise distances. That is, an assignment of coordinates is sought, such that the ratio $d_G(i,j)/d_E(i,j)$ between the graph distance d_G and the embedding distance d_E is approximately one for all node pairs $(i,j) \in G$.

To compute the similarity between two songs based on a graph demands for a costly shortest path calculation if the songs do not happen to be neighbors.[2] This shortest path evaluation does not only imply long calculation times but also exhibits an extremely high memory footprint. Having an embedding, the (Euclidean) distance between songs can directly be computed

[2]Observe that songs are typically not direct neighbors, as the graph needs to be sparse. Non-sparse graphs, with, say $\Theta(n^2)$ edges are too big to be stored.

from their coordinates, i.e. in O(1) time and with O(1) memory consumption. No information about any other songs or structures is required. Embeddings are thus particularly well suited for distributed and mobile applications. Moreover, an embedding exhibits several functional advantages, such as notion of direction, or the possibility to span volumes. A variety of interfaces that take advantage of these properties is presented in the next chapter.

Most state-of-the-art graph embedding algorithms are not well suited for large graphs. Already a complexity of O(n^2) exceeds memory or computation time limits. However, there exist methods that overcome these problems, such as the MIS-filtration algorithm of Gajer et al. [37], the high-dimensional embedding approach [43], or the landmark MDS algorithm (LMDS) [21].

We have decided to use LMDS, as it is not only fast[3] but also exhibits other appealing properties. LMDS works by first embedding a set of l – typically randomly selected – landmarks using classical MDS, and by then placing the remaining points according to their distances from the landmarks. Thus, the result closely resembles the widely used MDS embedding. Moreover, adjusting the number of landmarks follows a time-quality trade-off that allows to well adapt the resulting embedding to the application's needs. Finally, LMDS behaves well in dynamic settings. New nodes can be added to the embedding by placing them according to their distances to the landmark nodes, without changing the existing coordinates.[4]

To create the final map, we have improved on the basic LMDS algorithm by introducing the idea of *iterative embedding*, which will be discussed next.

9.2.1 Iterative Embedding

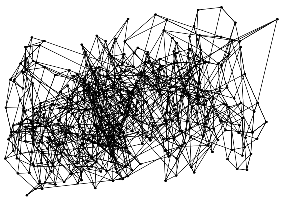
Iterative embedding is a major ingredient to the process of mapping our music graph into a Euclidean space. The idea is to successively improve the embedding by estimating the correctness of links using the coordinates calculated in the previous round. The technique applies to any sort of embedding algorithm. However, it assumes that the underlying data exhibits some randomness in its link structure, i.e., that some links erroneously shortcut certain paths. This property is generally attributed to small-world networks. Both the model of Watts and Strogatz [125] and the model of Kleinberg [56] for such graphs are based on this kind of edges. We expect the music graph to exhibit such characteristics, too, much like other naturally grown graphs, such as social networks, the WWW, or the graph of Wikipedia articles.

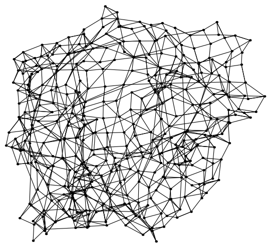
First the embedding algorithm is applied to the graph, resulting in a set of coordinates. Based on these coordinates, the fraction f of edges with

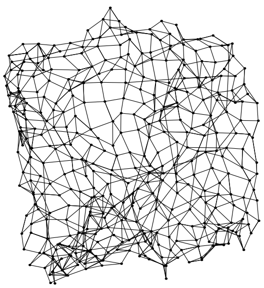
[3]The time complexity of LMDS is $O(nld + l^3)$, where n is the number of vertices, l the number of landmarks, and d the number of dimensions.

[4]This only works as long as the new nodes do not significantly affect existing graph distances.

maximum stress is removed. For our experiments we assumed that random edges get particularly long, meaning the stress increases as the ratio d_E/d_G increases (other stress functions might be defined for other settings). Identifying (and removing) this fraction f of edges can be done efficiently. Next, a new set of coordinates is calculated by embedding the graph after edge removal. This process is repeated k times, where k should be chosen such that $f \cdot k$ approximately matches the expected number of random edges. Moreover, f should not be chosen too large, as this might result in wrong edges being removed. The effect of iterative embedding ($f = 0.3\%$) on a 20×20 Kleinberg graph (grid augmented with random edges) is illustrated in Figure 9.1.


(a) The original embedding.


(b) After 6 rounds.


(c) After 12 rounds.

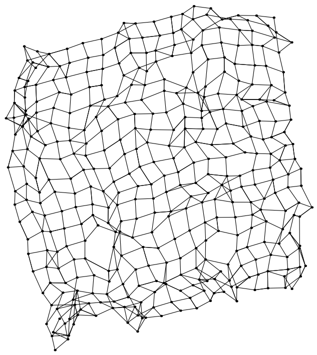

(d) After 30 rounds.

Figure 9.1: *Iterative Embedding* on the Kleinberg Graph: The original algorithm (a) cannot reconstruct the underlying grid. After 30 iterations, however, the grid structure can clearly be seen (f).

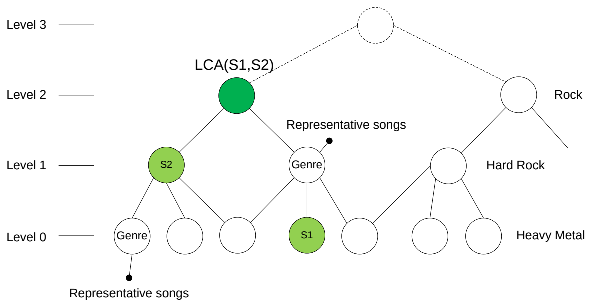


Figure 9.2: Music Taxonomy: Example of a genre hierarchy. The distance between two genres is the level of their LCA. E.g. two songs i and j belonging to genres $s1$ and $s2$ have genre distance $d_S(i, j) = 2$.

For the Kleinberg graph we used a spring embedding method (based on ideas from [121]) that is supposed to be well suited for small-world networks. However, the iterative approach is generic and works in conjunction with any embedding method. The effect of iterative embedding (using LMDS) on the music graph is illustrated in the following section.

9.2.2 Evaluation of the Resulting Map

To evaluate the quality of our embedding we used the genre information available at the *Allmusic Guide*. The *Allmusic Guide* is one of the largest music databases available on the web. As of 2007,[5] it provided a 3-level hierarchy of more than 700 music genres, as well as lists of "representative songs" assigned to each genre, as illustrated in Figure 9.2. This genre information is manually edited by community experts. We were able to match approximately 7000 songs between the *All Music Guide* dataset and our map. We used this subset of songs with known genre information in our experiments.

We define the distance d_S of two genres in this hierarchy as the level of their least common ancestor (LCA). Moreover, let H_s denote the height of the root in the corresponding tree. We can then use the hierarchy, together with this distance definition to define two quality measures:

(1) *Distance comparison* Q_L: The more distant two songs are in the genre hierarchy, the larger their distance should also be in the Euclidean space. Q_L thus summarizes the average similarity increase as a function of genre distance:

[5] I.e. at the time we conducted the experiments.

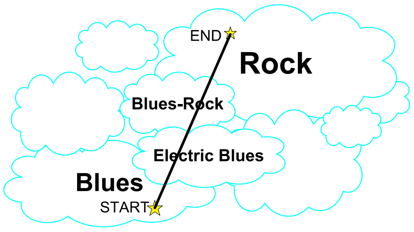


Figure 9.3: Embedding smoothness Q_R

$$Q_L = \frac{1}{H_S} \cdot \sum_{h \in \{0 \ldots H_S - 1\}} \left(\frac{\bar{d}_{h+1} - \bar{d}_h}{\bar{d}_h} \right), \tag{9.1}$$

where $\bar{d}_h$ is the average similarity of pairs of items (i, j) that have genre distance $d_S(i, j) = h$. We thereby assume that the similarity of unrelated songs is (typically) overestimated, such that higher values of Q_L indicate better quality.

(2) *Embedding smoothness* Q_R: In a good embedding, songs of the same genre should cluster. Moreover, for each genre there should exist only one cluster, and we expect these clusters to have convex shape. Therefore, a straight line between two random points (or songs) i and j in the embedding should reflect a gradual and systematic genre transition from i to j, as illustrated in Figure 9.3. An ideal gradual transition means that, once the line has crossed a cluster of a genre, it does not intersect the same or another cluster of this genre. The measure counts the number of violations of this rule, i.e. it counts how often an already visited genre re-occurs on a random straight line:

$$Q_R = \text{avg}(\#\textit{re-occurrences on a random line}). \tag{9.2}$$

To implement Q_R, the line between two random songs is sampled at 50 uniformly distributed points. To each of these points, the genre of the closest song is assigned.

We have used these quality measures to decide on the dimensionality of the output space, and to analyze the effect of iterative embedding.

The dimensionality of the target space significantly affects the quality of an embedding and follows a trade-off: The more dimensions, the lower the distortion of the embedding becomes. However, a larger number of dimensions implies higher memory and computing time requirements for an application that operates on the coordinates, and it also complicates the design

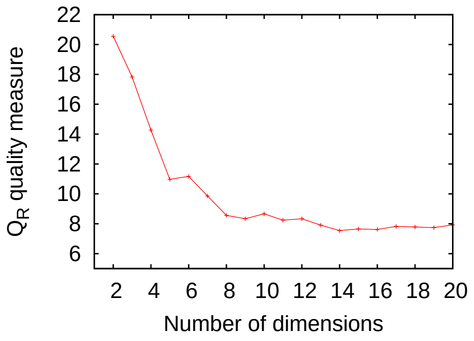


Figure 9.4: Improvement of the embedding smoothness Q_R with increasing number of dimensions.

of algorithms and interfaces. In an attempt to find the optimal number of dimensions for our purposes, we have used the embedding smoothness quality measure Q_R (defined in Equation 9.2), as it well matches the objectives of typical applications. Figure 9.4 reveals that increasing the dimensionality significantly improves the quality up to approximately 10 dimensions. This is in contrast to previous work that focuses on embedding in at most three dimensions and aims exclusively at visualization.

We have applied iterative embedding to the music graph in conjunction with LMDS and estimated the improvement over pure LMDS using the distance-comparison quality measure Q_L, defined in (9.1). The result for an 10-dimensional embedding on 430K nodes, with parameter $f = 0.5\%$, is illustrated in Figure 9.5. The figure shows a continuous quality improvement up to approximately iteration 30. We expect that at this point most of the random edges have been removed. Further iterations thus result in the removal of relevant edges and hence in a reduced embedding quality.

Figure 9.6 visualizes the main genre clusters of the embedding before and after 30 iterations. To be able to visualize the clusters, the embedding was projected into a 3-dimensional space (by taking the first three dimensions of each coordinate). The center of each cluster (ellipse) was placed at the center of mass of songs belonging to the corresponding genre. The ellipses' axes ("diameters") were set to half the value of standard deviation in each of the three dimensions. It can be seen that after the iterative embedding is applied, different clusters are better separated (even in three dimensions). In 9.6(a) one can see that the "Blues" and "Rock", as well as the "Jazz" and "World" clusters (and "R&B" and "Rap") are merged, whereas in 9.6(b)

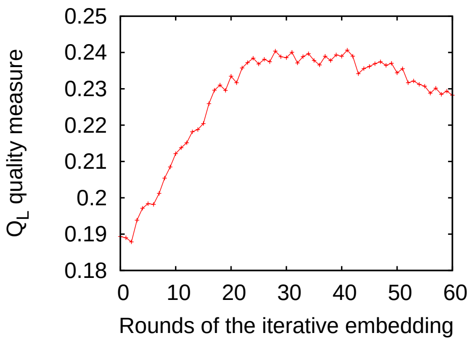


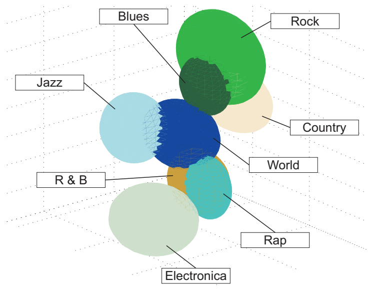
Figure 9.5: Quality improvement using iterative embedding.

these pairs of clusters are clearly separated. Although this visualization is limited due to dimensionality reduction, it serves as an additional indicator of the improvement achieved with iterative embedding. Moreover, it illustrates the good overall quality of the embedding.

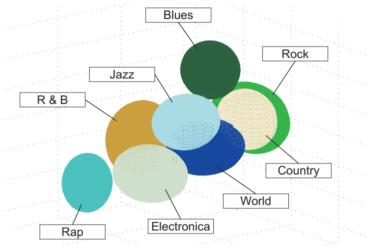
Note that the axes of the resulting Euclidean space are not assigned any semantical meaning, in contrast to the conventional notion of an axis being associated to some property, such as force or time. Table 9.1 illustrates the 10-neighborhood of two songs. The fact that the closest neighbors of each song belong to the same artist or to very similar artists shows that (1) the co-occurrence measure is in fact able to find similar items, and (2) that the step from usage based similarity information to a music map was successful, i.e. that the Euclidean map groups similar items together.

9.3 Method 2: Map Creation using PLSA

In the previous section we have shown how a map of music can be constructed from usage information using graph embedding techniques. In the following, we introduce another approach to derive a map of music from social data. This second approach is based on Probabilistic Latent Semantic Analysis (PLSA), which is a statistical framework to analyze co-occurrence data. The method was proposed by Hofmann [49, 50], and originally designed for automated document indexing. Similarly as in the widely known Latent Semantic Indexing (LSI) approach [22], the idea is to discover relationships between words and documents in a document collection. While LSI relies on techniques from linear algebra for this purpose, PLSA makes use of a probabilistic model. In particular, it introduces latent variables (also called latent



(a) Output of the original LMDS algorithm.



(b) After 30 rounds of iterative embedding.

Figure 9.6: Non-iterative LMDS (a) is unable to separate the genre clusters, whereas after 30 rounds of iterative embedding, the clusters become clearly disjoint.

classes) that interrelate words and documents. In a comparative study, Levy and Sandler [66] have found that PLSA provides better results than LSI in a context very similar to ours.

The goal of PLSA is to find probabilistic assignments between documents and latent classes, and between latent classes and words such that the observed occurrences of words in documents is best possible approximated by the probabilistic model. Hofmann shows that the corresponding assignments of documents to latent classes can be interpreted as a vector space. Thus, every document can be seen as a point in this space, and, as a result of the analysis, similar documents reside at similar locations in this space. To measure distances in the resulting space, the L_2-norm can be used.

PLSA has successfully been applied in fields other than document indexing, e.g. in the context of image databases, to construct a image similarity space. Analogously, a latent semantic music space can be constructed by considering songs as documents, and the social tags assigned to these songs as the words within the "documents" [66]. The result is a space of music in which similar songs are supposed to cluster.

9.3.1 PLSA

To describe the PLSA method we will make use of the notation used for document indexing, i.e. we will talk about *documents* d, *words* w, and *latent classes* z. We will thereby assume that there are N documents, M words, and K latent classes.

Pink Floyd (Time)	**Miles Davis (So What)**
Pink Floyd (On the Run)	Horace Silver (Song For My...)
Pink Floyd (Any Colour...)	Bill Evans (All of You)
Pink Floyd (The Great G...)	Miles Davis (Freddie Fre...)
Pink Floyd (Eclipse)	Nat King Cole (The More I...)
Pink Floyd (Us and Them)	Miles Davis (So Near)
Pink Floyd (Brain Damage)	Miles Davis (Flamenco Sk...)
Pink Floyd (Speak to Me)	Charles Mingus (Eat That Ch...)
Pink Floyd (Money)	Jimmy Smith (On the Sunny...)
Pink Floyd (Breathe)	Julie London (Daddy)
Pink Floyd (One of These...)	Bill Evans (My Man's Gone...)

Table 9.1: Closest neighbors of *Time* (Pink Floyd) and *So What* (Miles Davis)

In PLSA, documents are related to words via latent classes. Thereby, a generative model is assumed, in which a document is created by producing its words as follows: Each word is generated by first choosing a latent class, and then, dependent on the latent class, choosing a word. More precisely, for each word, first, a latent class is chosen with a certain probability $P(z_k|d_i)$. Dependent on this latent class, then, a word is chosen according to the probability $P(w_j|z_k)$. In this model, the probability that a document d_i creates a certain word w_j is given by:

$$P(w_j|d_i) = \sum_{k=1}^{K} P(w_j|z_k) \cdot P(z_k|d_i)$$

PLSA tries to find the assignment of the corresponding probabilities that best approximates the effectively observed document-word co-occurrences. The optimization of the model parameters (i.e. the probabilities) is done using the well known Expectation Maximization (EM) technique that works by alternately applying an expectation (E) and a maximization (M) step. The iterations are aborted as soon as some convergence criterion is met (between 20 and 50 iterations have shown to sufficient in practice).

In the context of PLSA, the goal is to maximize the likelihood L' of the observed data:

$$L' = \prod_{i,j} P(d_i, w_j)^{n(d_i,w_j)} = \prod_{i,j} \left(\sum_k P(w_j|z_k) \cdot P(z_k|d_i) \right)^{n(d_i,w_j)}$$

In practice, it is more convenient to work with the logarithm of the likelihood, which has its maximum at the same position. The log-likelihood L follows from the above equation:

$$L = \sum_i \sum_j n(d_i, w_j) \cdot \log P(d_i, w_j) \tag{9.3}$$

The corresponding expectation and maximization steps are given below:

- Expectation step:

$$P(z_k|d_i, w_j) = \frac{P(w_j|z_k) \cdot P(z_k|d_i)}{\sum_{l=1}^{K} P(w_j|z_l) \cdot P(z_l|d_i)} \tag{9.4}$$

- Maximization step:

$$P(w_j|z_k) = \frac{\sum_{i=1}^{N} n(d_i, w_j) \cdot P(z_k|d_i, w_j)}{\sum_{m=1}^{M} \sum_{i=1}^{N} n(d_i, w_j) \cdot P(z_k|d_i, w_m)} \tag{9.5}$$

$$P(z_k|d_i) = \frac{\sum_{j=1}^{N} n(d_i, w_j) \cdot P(z_k|d_i, w_j)}{n(d_i)} \tag{9.6}$$

9.3.2 Applying PLSA to Music

To create our social audio features, we have applied the PLSA method to data gathered from *last.fm*. Thereby, we consider two sources of information: (1) Social tags, such as they were assigned by users to songs and (2) the listening behavior of the users, as given by the *top-50 lists* that were already used to create the graph embedding based map. The following discussion is based on crawled data from about 2.4M users, containing approximately 10M songs, 1.4M artists, and 1M tags.

Observe that this information could be used in different ways in conjunction with PLSA, which basically only requires some sort of co-occurrence data:

- Using user-song co-occurrences, similarly as this was done in the graph embedding based map.
- Using the co-occurrence of songs and social tags (as described in [66]).

Using the song-tag co-occurrences is an intuitive approach and has proven to work well. We improve upon this basic approach by smartly re-assigning tags prior to applying PLSA. Thereby, we implicitly take advantage of the user-song co-occurrences, and thus effectively combine the two information

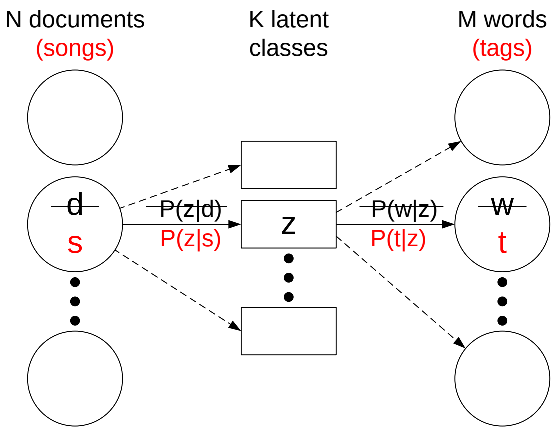


Figure 9.7: Principle of the Probabilistic Latent Semantic Analysis applied to songs and tags.

sources. In the evaluation part, we will show that this combination leads to a significant performance gain as compared to approaches that only consider a single information source.

In the following, songs are considered as documents and (re-assigned) tags are considered as words in the context of PLSA. Thus, for the remainder of this section, we will change our notation as follows:

- document → song: $d \rightarrow s$
- word → tag: $w \rightarrow t$

This shift is also illustrated in Figure 9.7.

The tags assigned by *last.fm* users exhibit some peculiarities. In particular, there are lots of *personal tags* that relate to the user who assigned them rather than to the music. Examples are “heard on pandora”, “favorite artist”, and “awesome”. Moreover, several spellings are used to denote the same thing, such as “hip hop”, “hip-hop”, and “hiphop”. To reduce noise, we only considered the approximately 1K most occurring tags and manually cleaned them by removing *personal tags* and by normalizing synonyms. The reduction to the most occurring tags on the one hand allows for manual cleaning, and on the other hand considerably reduces the computational complexity without significantly affecting the accuracy.

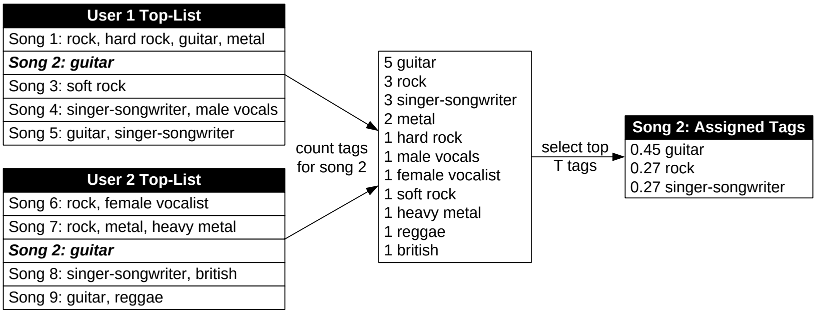


Figure 9.8: Assigning weighted tags to a song (simplified example with only 2 top-lists).

Another issue with social tags is that they suffer from a popularity bias as shown by Turnbull et al. [117]. That is, typically only the most famous songs are accurately tagged, whereas unpopular songs often contain no, or inappropriate tags. In fact only about 20% of the songs in our last.fm subset were tagged by the users. Directly applying PLSA to the song-tag co-occurrence data, such as done in [66], would thus exclude the remaining 80% of the songs, which is not acceptable for the use in real-world applications.

To overcome this problem, we make use of the information contained in the users' top-50 lists. Similarly as in item-to-item collaborative filtering [71] we assume that songs that often occur together in such top-lists are related to each other to a certain degree. This assumption facilitates the extrapolation of the tagging information to previously untagged songs. In Section 9.3.3 we will show that this does not only solve the data sparsity problem, but also improves the accuracy of the resulting space.

In particular, we automatically assign tags to a given song s using the following procedure: For each top-list s appears in, iterate through all the neighbors (i.e. through all the other songs in the corresponding top-list). For each neighbor, iterate through all its tags (i.e. all the tags that last.fm users have assigned to this particular song). For each of these tags, increase the song-tag co-occurrence value of song s. Finally, assign the T tags with highest co-occurrence to song s for the use in the PLSA optimization (for some threshold T, 50 in our case). Moreover, these tags are weighted proportionally to their occurrence numbers (and such that the weights sum up to 1). This automated tag assignment process is illustrated in Figure 9.8 for a simplified example with only two top-lists (and $T = 3$).

We have tagged the (approximately) 1.1M most occurring songs using the described technique. The obtained song-tag co-occurrence data has then been fed into the PLSA framework. After applying PLSA, the conditional probabilities $P(z_k|s_i)$ are well defined for all classes z_k and all songs s_i. These probabilities can be seen as coordinates, assigning each document a point in the so called *probabilistic latent semantic space* [50]. Since the probabilities corresponding to a song s_i sum up to 1 (i.e. $\sum_k P(z_k|s_i) = 1$), the songs in fact lie on a $K - 1$ dimensional hyperplane. The corresponding latent semantic space forms a map of music – the PLSA map.

The resulting space covers roughly 1.1M songs corresponding to more than 120K artists. However, our *last.fm* database contains information to more than 1.4M artists. To make this information available we have calculated *artist coordinates* for (most of) these artists as follows:

- For all the artists that are available in the latent space, we define the *artist coordinate* as the center of mass of their songs with known coordinates.

- For the remaining artists we have queried their closest neighbors from *last.fm*. We then place the artist at the center of mass of all the neighbors with known coordinates (as calculated before). This procedure was successful for about 1M artists.

As a result we could define coordinates for more than 1.1M artists, which is enough to facilitate the use in end-user applications.

An advantage of the PLSA-map over the LMDS-map is the direct relationship to the tags that were used in its construction process. In particular, the PLSA-model inherently defines the probability of a song generating a given tag as:

$$P(t_j|s_i) = \sum_k P(z_k|s_i) \cdot P(t_j|z_k) \tag{9.7}$$

We will make use of this relationship in the mobile application presented in the very end of this thesis.

9.3.3 Evaluation

In this section we describe different tests that allow to compare the qualities of different music similarity spaces. In particular, we will compare the spaces resulting from the following approaches:

- *Song-tag approach*: PLSA is applied to plain song-tag co-occurrences (as given from last.fm), i.e. the same information as used in [66].

- *Song-user approach*: PLSA is applied to song-user co-occurrences, i.e. the same information as used for the LMDS map.

- *Combined approach*: PLSA is applied to the co-occurrences of songs and re-assigned tags, as described before.

Since the song-tag approach suffers from the mentioned popularity bias problem, the comparison is done on a reduced dataset. Similarly as in [66] we only consider songs that contain at least 30 tags. To ensure fairness with respect to the *song-user approach*, we have also eliminated songs that appear in less than 30 top-50 lists. The resulting *reduced dataset* contains roughly 80K songs. To construct the space with the *combined approach* we have applied PLSA to both, the reduced as well as the full dataset. To keep the numbers comparable, the same 80K songs were used for both datasets during evaluation.

Our tests are based on three different criteria to assess the quality of a given music space:

- *Consistency of social tags*: In a space that well reflects perceived music similarity, the social tags of songs in a close neighborhood should be similar.

- *Comparison to collaborative filtering*: Collecting a sufficient amount of human judgments to get a *ground truth* with respect to perceived similarity is an extremely expensive task. Moreover, there do not seem to be any publicly available datasets that can be used for this purpose.[6] Thus, we rely on item-to-item collaborative filtering as a "ground truth" to which we can compare our results. Recall that Berenzweig et al. [13] have compared a variety of music similarity measures and found that item-to-item collaborative filtering performs best among the investigated approaches.

- *Artist clustering*: Songs of the same artist are often similar. Thus, songs of the same artists are supposed to (somewhat) cluster in a space that well reflects music similarity.

In Section 9.4 we will use the same measures to compare the PLSA map (*combined approach*) to the LMDS map.

[6]Unfortunately, the datasets developed in the context of the MIREX project [25] do only seem to be available when participating in one of MIREX tasks.

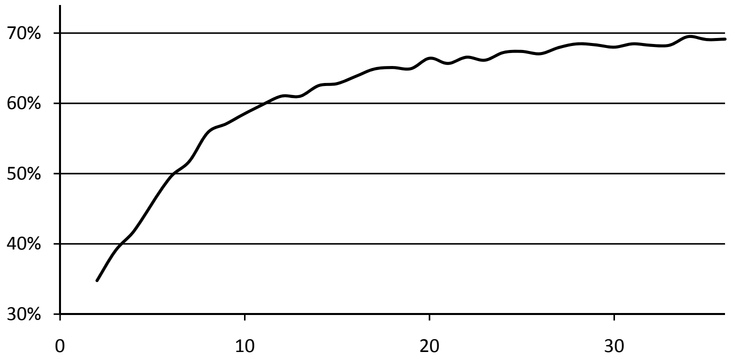


Figure 9.9: The number of latent classes versus the percentage of correct tag estimations. The plot shows that increasing the number of latent classes to more than about 30 does not lead to a relevant quality improvement.[8]

Tag Consistency

To evaluate tag consistency, we try to estimate the (uncleaned) tags of a given song by considering the (uncleaned) tags of songs residing in its neighborhood. For this purpose, we closely follow the concept of a k-nearest-neighbor (KNN) classifier. To estimate the tags of a song s, we consider the tags assigned to the k closest songs in the music space (for $k = 20$). For each tag we count the total number of occurrences. The 10 most occurring tags are then compared to the 10 tags that were most often assigned to s by *last.fm* users. The percentage of correctly estimated tags is a good measure to compare different music spaces to each other. We have not only used it to compare the *combined approach* to the other two PLSA variants, but also to decide on an appropriate number of latent classes (i.e. the dimensionality of the resulting latent semantic space). Figure 9.9 plots the number of latent classes versus the percentage of correctly estimated tags. As expected, the number of latent classes significantly influences the accuracy of the resulting music space. An important observation is that the curve levels off. That is, increasing the number of dimensions beyond about 30 does not lead so a significant increase in the precision of the estimated tags. Thus, we have fixed the number of dimensions to 32.

We have measured the tag consistency on the three PLSA variants (song-tag, song-user, and combined) on the reduced dataset, and, in addition, on the combined variant on the full dataset. The results are summarized in

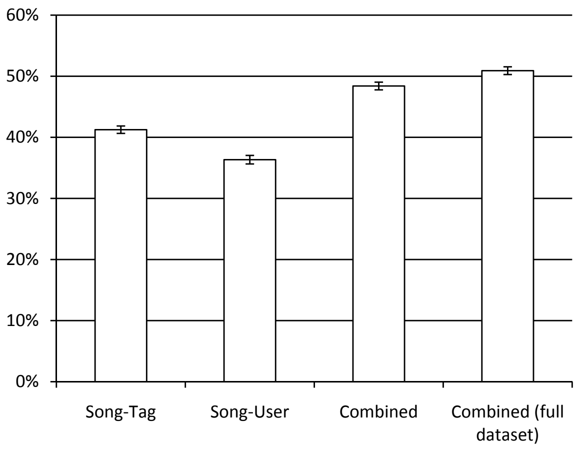


Figure 9.10: Tag consistency: Combining the two information sources (social tags and listening behavior) improves the KNN tag estimation.

Figure 9.10. On the one hand, we can see that the combined variant significantly improves the tag estimations as compared to the simple approaches. On the other hand, the figure shows that on the full dataset the performance even increases. Over half of the most relevant user-assigned tags could be correctly estimated by looking at the neighborhood of a song – a remarkable number when considering the synonym issues and the many personal tags present in the last.fm dataset.

Comparison to Collaborative Filtering

The *top-50 lists* available from our last.fm dataset can be used to calculate a ranked neighborhood of a song using item-to-item collaborative filtering (our "ground truth"). Collaborative filtering is designed to identify the most similar items and is known to perform well in this respect. However, for the majority of distant items, it does not provide any information due to the lack of co-occurrence information. In our experiments, we thus compare the k closest neighbors in our map with the k most similar items identified by item-to-item collaborative filtering, applied to our *top-50 lists*.

The results of applying this measure to the different space construction variants are shown in Figure 9.11. Again, we can see how the combination of the two signals significantly improves the quality. The approximately 25% matches in the 10 closest neighbors should be contrasted to the 80K

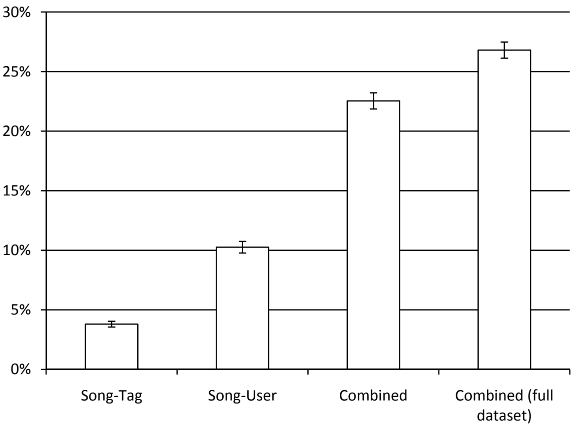


Figure 9.11: Comparision to collaborative filtering: The combined approach clearly outperforms the alternative approaches. More than 25% agreement is a remarkable number, considering that only 10 songs were compared from a universe of 80K songs.

songs these neighbors could be chosen from. The task is comparable to the search for a needle in a haystack – thus, more than 25% identical output is a remarkable result.

Artist Clustering

We measure the level of artist clustering using the mean average precision (mAP) on artist labels. Average precision (AP) is a standard performance metric known from information retrieval. It is used to measure the quality of a ranked sequence of items, such as given when ordering songs according to their distance from a given query song s. Thereby, relevant items (songs featuring the same artist as the query song, in our case) that appear early in the list are rewarded more than those that appear towards the end. More formally, AP is defined as

$$AP = \frac{\sum_{r=1}^{N} P(r) \cdot \mathrm{rel}(r)}{R},$$

where $P(r)$ denotes the precision at rank r, rel(r) is 1 if the item at rank r

[8]The absolute numbers cannot directly be compared to the other experiments, as this plot is based on a different dataset and different parameter settings.

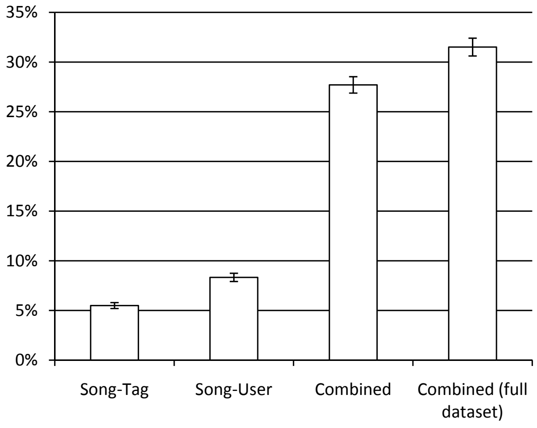


Figure 9.12: Artist clustering: Combining the two information sources significantly improves the mean average artist precision. Observe that this test is based on a total of 11K artist labels, which inherently leads to relatively low numbers.

is relevant (and 0 otherwise), R is the total number of relevant items, and N is the total number of retrieved items (i.e. all the songs, in our case).

Higher AP (and thus also mAP) values refer to better artist clustering. However, a better artist clustering does not necessarily imply a better quality of the underlying space, as there is no reason, why songs of other artists cannot be similar to the query song. *John Lennon* and *Beatles* might serve as an example. In the same way, a single artist can have songs of extremely different style (e.g. *Nothing else Matters* and *Master of Puppets* from *Metallica*). Thus the mAP performance metric is questionable in our context, in particular when comparing relatively high mAP values. As it has been used before to quantify the accuracy of music similarity (see, e.g. [66]), we will apply it as well, despite its questionable nature.

The corresponding results are summarized in Figure 9.12. In line with the previous results, the figure shows that the two information sources (listening behavior and tags) can profit from each other. And again, the result of the combined approach even improves for the full dataset. When comparing these numbers to other approaches, it is important to consider the number of artists in the dataset. In [66], for example, the dataset contained 212 artists, as opposed to roughly 11K in our experiments (which inherently leads to lower mAP values).

9.3.4 Computational Complexity

The EM equations (Equations 9.4-9.6) suggest that the computational complexity of a single PLSA iteration (consisting of one estimation and one maximization step) is O(KMN). However, the matrices are typically sparse, which results in a reduced complexity if an appropriate implementation is used. Most documents contain only a small subset of the possible vocabulary, resulting in a sparse document-word co-occurrence matrix (such as in our case, where each song is automatically tagged with 50 tags). Thus, most values for $n(d_i, w_j)$ in Equations 9.5 and 9.6 are zero. Using appropriate data structures, the zero-elements of this matrix do never have to be considered, such that the computational complexity reduces to O(KC), where C is the number of entries in the co-occurrence matrix. Moreover, Hofmann [50] states that typically around $20 - 50$ iterations (i.e. a constant number) are sufficient, such that the overall complexity does not significantly increase.

In fact, we have analyzed the required number of iterations in our scenario. Hofmann proposes to measure the log-likelihood on held-out data for this purpose, and to stop iterating as soon as the log-likelihood starts to decrease, which should prevent overfitting [50]. Later studies, however, have shown that overfitting is not a big issue with PLSA [16], which renders the use of held-out data unnecessary. We have thus decided to simply measure the log-likelihood within the entire dataset. Figure 9.13 plots the log-likelihood versus the number of iterations. We can see that after a significant increase after roughly 10 iterations, the log-likelihood starts to level off. It is interesting to compare these results to Figure 9.14, which plots the *tag consistency measure* versus the number of iterations. We can see a similar behavior, which shows that the PLSA model and the corresponding log-likelihood well describes the semantics of music. Moreover, this result confirms that PLSA (at least in our setting) is robust with respect to overfitting. For performance reasons (and simplicity), we in practice stop the expectation maximization after 50 iterations. This number lies well within the observed convergence region and agrees with Hofmann's statement.

9.4 Comparison of the two Maps

To compare the two maps, we rely on the criteria introduced in Section 9.3.3. The corresponding numbers are listed in Table 9.2.

The results show that the two maps score almost equally with respect to tag consistency. In the comparison to collaborative filtering, and with respect to the mean average artist precision, the LMDS map (Method 1) outperforms the PLSA map (Method 2). The high correspondence between a song's closest neighbors in the LMDS map and in the item-to-item collaborative filtering

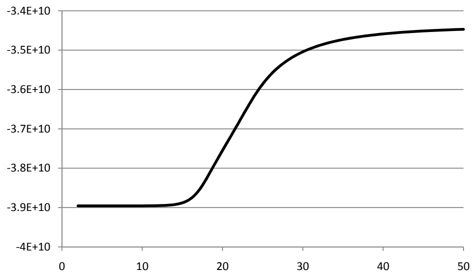


Figure 9.13: The number of iterations versus the log-likelihood given by the PLSA model.

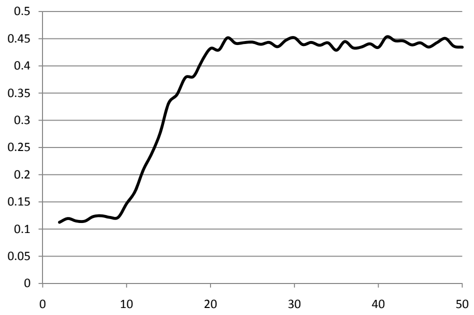


Figure 9.14: The number of iterations versus the tag consistency measure.[9]

Measure	LMDS	PLSA
Tag Consistency	53%	51%
Comparison to CF	57%	27%
Artist Clustering	59%	32%

Table 9.2: Quality comparison of the two maps according to different criteria.

neighborhood shows that the relevant information could be retained in the graph embedding step. (Recall that the LMDS approach makes use of graph embedding to map the collaborative filtering based similarity measure into Euclidean space.) Interestingly, the PLSA-based map does not score better with respect to tag consistency, even though it uses the tag information as a direct input. Overall, the numbers indicate that the LMDS map is preferable in typical application scenarios. However, at the time of writing, the PLSA map exhibits two advantageous properties: It covers a bigger number of songs (and artists), and it compactly defines the probability of a certain tag being created by a given song (or point in space). For these reasons, we used the PLSA map for the Android application presented in the next chapter, despite its slightly lower quality compared to the LMDS map.[10]

To conclude this chapter, we want to quickly review some major properties of the generated music similarity maps:

- *Similarity measure*: For both maps, the underlying similarity measure is based on an analysis of *last.fm* data. In the embedding based map (Method 1) the actual similarity calculation solely relies on the users' listening behavior and closely resembles Amazon's item-to-item collaborative filtering idea [71]. In the PLSA based map (Method 2), the similarity measure combines information from social tags and the user's listening behavior.

- *Dimensionality*: The dimensionality analysis of both construction methods has shown that an intuitively navigable 2 or 3-dimensional space is not able to adequately reflect music similarity. Moreover, for both methods we could identify a point at which further increasing dimensionality does no longer significantly affect the accuracy. The resulting dimensionality of the two spaces was chosen accordingly: 10 dimensions for Method 1, and 32 dimensions for Method 2.

- *Orientation*: The first space merely represents relative similarity of songs. Axes in this space are not assigned a special meaning, which makes it hard to guide the user through the space. In the second space, based on PLSA, the axes are defined by latent classes which are directly related to the underlying tags. This information can be helpful to guide the user through the space.

[9]Again, the numbers are not directly comparable to the other experiments, as they are based on a slightly different dataset and different parameters.

[10]It seems that both issues could be overcome when recreating an LMDS map with our latest *last.fm* dataset, however, memory optimized code would be required to create such a map of comparable size, and additional tests would be required to show the quality gains.

- *Coverage*: The LMDS based space comprises approximately 430K tracks, which covers about 50%-60% of a typical music collection – enough to conduct meaningful experiments, but not enough to build an end-user application. The PLSA based space contains more that 1M tracks corresponding to more than 120K artists. To further increase the coverage, we have calculated the coordinates of an additional 1.3M artists. With that, the coverage gets high enough to be usable in productive applications, such as the one shown in the end of the thesis. The major reason for the better coverage of the PLSA space is that it was constructed later, when we possessed a larger subset of *last.fm* data.

9.5 The Maps' Properties

Working on a Euclidean space rather than on pairwise distances exhibits several advantages. On the one hand, applications can benefit from the geometric properties of such a space. On the other hand, the compact representation using coordinates saves memory resources and computing time, and it allows for use in distributed applications. To conclude the chapter, we briefly discuss these different aspects.

Geometry The geometric properties of Euclidean spaces facilitates the realization of new functionality. In particular, applications can profit from properties such as *trajectories*, *volumes* and the *notion of direction*. These elements constitute the building blocks of an application. Our music maps are designed such that similar songs are grouped together. Therefore, regions in space can typically be associated with certain music properties, such as genre, mood, rhythm, and so on. When mapping a user's favorite songs to one of our music maps, the region(s) they occupy can be compactly represented as a volume (or a union of several volumes). Hence, we can, for example, use a volume to compactly define the *region of interest* of a user. Trajectories, on the other hand, allow to smoothly *interpolate* between songs or regions. In Chapter 10 we will see an example that takes advantage of this property to create smart playlists by defining a start and an end song. Finally, using sense of direction allows to *extrapolate* such trajectories. Given a sequence of songs, we can define how this list could be extended.

Memory and Computation Time Representing music similarity in a space, rather than by means of a graph resulting from pairwise distances, exhibits several advantages in terms of resources. In a Euclidean space the similarity of two songs can efficiently be evaluated by comparing their coordinates. In a graph, the same operation requires the computation of the shortest path between the songs. One might argue that (approximate) shortest paths can sometimes be calculated quickly, by means of the A*-algorithm [44], or approximate distance oracles (see, e.g. [114]), for example. However, on large graphs these operations are still considerably slower than the computation of the Euclidean distance from two points of comparably low dimensionality (≤ 32 in our case). Moreover, there is an inherent problem about memory consumption, when working on the raw graphs. Observe that storing the similarity information of the entire music universe can be done in $O(n)$ space in the case of a Euclidean representation, as opposed to $O(m)$ space for a graph representation (where n denotes the number of songs, and m the number of edges in a graph). Again, one might argue that using constant bounded degree graphs, $m = O(n)$, such that there is not relevant difference. However, in practice such a degree bounding constant would need to be large in order to guarantee a good similarity representation as well as the connectivity of the graph. Moreover, many applications do not need to know about the similarity relationships among *all* the songs in the music universe. Rather, applications often operate on a small subset of this universe, such as on a particular user's music collection. Using our map representation, we only need to store the coordinates of the relevant songs on the target device, in order to make the corresponding similarity information available. Using pairwise similarities, by contrast, we also need to store all the intermediate songs such that distances can be defined transitively (i.e. by means of shortest paths in graphs). The obtained savings in terms of memory consumption are typically immense, considering that collection sizes are typically in the order of thousands of songs, as opposed to hundreds of thousands of songs contained in our music maps. The outlined advantages in terms of computing time and memory requirements make a map representation the perfect candidate for use on personal collections in the mobile domain.

Distributed Systems As discussed, a set of songs (such as a personal music collection) can compactly be represented by the coordinates of the corresponding songs. Moreover, two sets of songs (or collections) can easily be compared by looking at these coordinates. Observe that for such a comparison, no global database is required. As a consequence, the information can be well used in distributed settings. Consider, for example, a smart decentralized file-sharing system: Assume that a user's mobile phone (containing a music collection) detects another mobile device in its vicinity (by means of Bluetooth, for example). The two devices could exchange the coordinate information of their collections, possibly in a compact representation as a volume. Then, they could identify regions of common interest (i.e. volume intersections) and start exchanging songs that fall into these regions and are not yet present on both devices. Using such a system, a user's device could smartly collect songs of interest while its owner, say, takes a walk through the city.[11] In a similar way, mobile devices could help to find like-minded people (in terms of music taste) in a mobile match-making application, such as the previously presented VENETA system. These examples should just demonstrate one of the many advantages of a the representation of music similarity in a Euclidean space. We did not realize such a distributed application. Rather, we focused on the implementation of smart interfaces for music access on mobile devices, as we will see in the next chapter.

[11]Clearly, several issues concerning intellectual property rights need to be resolved before releasing such a system. Thus we did not implement it. Here we just wanted to sketch the technical possibility.

Chapter 10

The Map of Music in the Mobile Domain

Over the past years, mobile phones have turned into multi-purpose entertaining devices with increasing storage capacity and ever better audio codecs for high-quality music reproduction. Ever larger collections of music are stored on mobile devices, making the process of managing these repositories more challenging.

Unfortunately, the interfaces offered by mobile media players lag behind this trend of ever growing collections. Today, the organization of digital music libraries is mostly handled with meta data included in the audio files. Traditional list based search and browsing options render it hard to keep an overview over a large amount of tracks. As a result, users experience problems selecting the appropriate music for a given mood or situation. Research about the users' needs in music information retrieval has shown that people are searching for music not only by means of bibliographic data (i.e. artist and title), but also in more descriptive ways, such as by specifying genre or mood information, or by naming artists that are similar to the desired music [9, 26, 64]. However, current implementations of digital music players do typically not support unspecific search queries like: "I want to listen to music similar to the current track, but not of the same artist", or "I would like to listen to something happy". Rather, the organization of the collection is handled with meta data included in the audio files. As we have discussed in Chapter 9, methods based on *music similarity* offer an alternative to traditional keyword based searches.

The demand for novel music management tools has been captured by many commercial projects. Online music communities, such as *last.fm*, and *iLike* have been offering novel ways to discover and experience music. By managing millions of user profiles, they apply collaborative filtering techniques to search for music, generate playlists, and recommend similar music. Most of these technologies are based on large centralized databases and thus not suited for mobile environments.

The concept of a music map as presented in Chapter 9 facilitates the use of such community based similarity measures in the mobile domain. It is a perfect tool for the organization of large collections on resource restricted devices. On the one hand, it serves as a basis for intuitive navigation, and on the other hand it enables an efficient computation various aspects related to music similarity. In order to compute the similarity between two songs, for example, it is enough to calculate the distance between the coordinates assigned to them. No memory-expensive data structures need to be maintained on the mobile device and no access to a centralized server is required.

However, we have also seen that the underlying similarity measure is intrinsically high-dimensional.[1] In the following we investigate several approaches to deal with such a high-dimensional space of music on a mobile device. We address the problem from different perspectives. First, we look at simple textual interfaces that consider similarity information to guide the user through a music collection. We then show a visualization technique that maps the high-dimensional data into the (2-dimensional) display space dependent on the area of interest. For this purpose we introduce a lens metaphor that allows to focus on one part of the collection but at the same time retains a global overview. Next, we present an "acoustic" interface that we call smart shuffling. The idea of this method is to keep track of the user's behavior, thereby allowing to avoid undesired regions of music. The algorithm tries to stay as broad as possible to make sure no areas that match the user's taste are missed. Finally, we present a visualization scheme that combines the advantages of the well known Cover Flow interface with those of similarity maps. The scheme comes in a 2D and a 3D flavor. For dimensionality reduction we take advantage of the fact that a user's collection typically does not contain all the similarity dependencies relevant to music. Thus, a reasonable low dimensional representation can be calculated on a collection basis using the high-dimensional coordinates as a starting point. This visualization scheme has been incorporated into a comprehensive mobile music application (*museek*[2]). The application also takes advantage of the tag information obtained from the PLSA based map of music. In combination with the map, these tags enable a fine-grained selection of music corresponding to a certain mood or style.

All interfaces are designed such that users can quickly find music that

[1]Under high-dimensional we understand everything that exceeds the intuitively visualizable number of 2 or 3 dimensions.

[2]`http://www.museek.ethz.ch`

matches their taste, even within collections they are partly or entirely unfamiliar with. Moreover, the methods attempt to give an overview of the entire collection, such that tracks from various music areas can serendipitously be discovered.

The usability of our methods has been evaluated in different user studies. We compare the lens-metaphor based visualization scheme and the smart shuffling algorithm to related interfaces in an experiment comprising 9 participants. In these experiments our visual interface outperforms the recently introduced SensMe feature of Sony Ericsson, which, to the best of our knowledge, is the only currently available commercial system that heads into a similar direction. Due to the lack of commercial mobile solutions in the area, we compare the smart shuffling approach to the widely used concept of random shuffling as well as to an algorithm proposed by Pampalk et al. [95] that aims at playlist generation based on an audio-feature space. In the conducted experiments, our algorithm scores better than the two alternative approaches (random shuffling and Pampalk's algorithm).

Finally, we present an analysis of the usage of the interfaces available in *museek*. The analysis is based on more than 100 log files that report the natural use of the application. The study stresses the need for advanced music retrieval interfaces. In line with studies about the user behavior in music web forums, we find that, while traditional search facilities remain important, less specific and more explorative ways to access music are frequently used and highly accepted by the users.

10.1 Related Work

The exploration of music collections by means of sophisticated interfaces is an active area of research. When designing such interfaces, or even entire music retrieval systems, it is essential to know about the needs of the end users: How do people search for music? What tools could assist them to find what they are looking for? To get a better understanding of the users' needs, Bainbridge et al. [9] have analyzed music queries in the *Google Answers* service. Not surprisingly, artist and song title are most often used to search or ask for music. However, the study also shows that roughly a third of the queries included a description of the genre or style, and that sometimes references to known similar musicians were given. Similar results were reported by Downie and Cunningham [26] in an study on newsgroup messages. Lee et al. [64] used questionnaires to investigate how people are searching for music. Their study emphasizes the importance of non-specific queries. Roughly 60% of the participants replied that they would make use of style information or similar artists to search for music, and also criteria such as popularity, mood, language, or vocal range and gender were shown to be relevant. Genre information is doubtlessly important in the context of music information retrieval. However, there is only little agreement on genre assignments and taxonomies [6, 61], which limits the usefulness of genres. An alternative to genres is given by social tags that provide several advantages [61]: They overcome synonymy issues (e.g. E-Jazz vs. Electronic Jazz), allow for a fuzzy assignment (a song might, e.g., be tagged with Nu-Jazz and E-Jazz), and can also grasp mood and other non-genre related information. Bentley et al. [12], finally, conclude that music retrieval systems could profit from support for serendipitous browsing capabilities. In the following sections we propose a variety of music retrieval interfaces that try to address the above findings. In particular, our interfaces provide the users with the possibility to search for music based on similarity information, and to browse music collections in a serendipitous fashion, such that they can (re-)discover music that matches their taste. We thereby cover a wide spectrum of techniques, ranging from simple textual interfaces to collection visualization and implicit interaction. Before we present our methods, we will briefly review some existing literature in the field.

The biggest class of existing interfaces tries to visualize collections based on previously extracted audio features. A popular approach is the use of self-organizing maps (SOMs) to create pleasant drawings of the underlying space. Mörchen et al. [81] proposed to apply emergent SOMs (ESOMs) to a complex audio-feature space to retain as much information as possible in the 2-dimensional output space. A 3-dimensional visualization of relatively small collections (50 songs), also by means of SOMs, is described in [58]. A spherical SOM has been used by Leitich and Topf [65] to create the globe of music. The PocketSOMPlayer [83, 35] applies SOMs for visualizations on small devices. All these approaches map the audio space into some low (2 or 3) dimensional representation, which can then be explored by traditional navigation schemes.

Besides self-organizing maps, various other techniques have been used to present the complex structures behind music similarity on a 2-dimensional display. Donaldson and Knopke, for example, apply a special node repulsion technique to overcome occlusion effects. The commercial SensMe interface of Sony Ericsson, displays songs along two axes (mood and tempo). How this information is extracted remains the company's secret. A more detailed discussion of SensMe is given in Section 10.3.

Moreover, several more abstract visual interfaces have been proposed to overcome the inherent problems of the low dimensional output space. The artist map proposed by van Gulik et al. [122] uses a spring embedder to visualize a collection along a user definable set of properties, which are partly extracted from the audio content and partly consist of meta-data. The approach is directed at small devices and restricted to artist similarity. A circular layout has been chosen by MusicRainbow [94] as well as AudioRadar [46], which both operate on an audio-feature space (in case of MusicRainbow augmented by keywords retrieved from the web). The interface proposed by Torrens et al. [115] does not rely on any music similarity measure. Rather, it directly visualizes meta data, such as genre or year of release, in a circular, a rectangular, and a tree-based fashion. Finally, we want to mention Apple's Cover Flow interface that merely replaces the traditional textual album list by nicely presented album covers. The enormous popularity indicates that album covers are a useful visual hint to retrieve music.

The visualization of high-dimensional structures has also been studied outside of the context of music. A comprehensive overview is given by [107]. Not all the described techniques can be applied to our setting, though, such as the rank-by-feature approach [106], which depends on known meanings of the axes. Two major ingredients often seen in high-dimensional data visualization are hierarchical decompositions (e.g. used in [127]) as well as icon based techniques (e.g. applied in [47]). One of the visual interfaces we propose combines these two techniques with a generalization of the fish-eye [103] view for higher dimensions.

Often, visual interfaces are combined with intelligent playlist generation. Examples are the commercial SensMe interface, the approach of van Gulik et al. [120], and the PoketSOMPlayer [83, 35] that allows to create playlists by drawing trajectories through a SOM-based map. We propose a purely textual interface, which lets the user pick a start and an end song and thereby also realizes the idea of trajectory based playlists. A quite different concept to access music is proposed by the MusicCube device [2] and the work of Bergman et al. [14] that both assign certain songs to a (virtual) 3D-location in the device and consider the device orientation to retrieve songs. Moreover, there are interfaces that augment the visual space with an acoustic environment, such as in [75] and [118]. Both of these interfaces try to take advantage of the *cocktail party effect*. That is, they play different pieces of music simultaneously. At the same time, they guide the user through the music space by means of visual aids.

A purely acoustic exploration method has been proposed by Pampalk et al. [95]. The goal is to find music that matches the user's taste by considering feedback such as skipping behavior. Even though their algorithm has been described for audio-feature spaces, it can also be used in combination with our map. We will later present an algorithm that follows a similar idea and, at least in the context of our map, performs better than Pampalk's algorithm. Other approaches that go into a similar direction, but are geared at the large listening community of a web radio, are presented in [45] and [8]. External context, rather than skipping behavior, is considered in the music selection process of the XPOD device [24].

A purely textual interface to generate intelligent playlists is Apple's iTunes Genius feature. It basically selects songs similar to the preceding songs and thus follows a similar principle as approaches presented in [73], [98], [5], and [99]. In museek we have implemented a play mode that realizes this idea.

In the following, we present a set of user interfaces that build upon our music maps. While all of the interfaces have been implemented for the Android mobile platform, most of them have *not* been included in museek. The interfaces and experiments presented in Sections 10.2 to 10.4 are based on the LMDS map. However, all of them could also be realized in conjunction with the PLSA map.

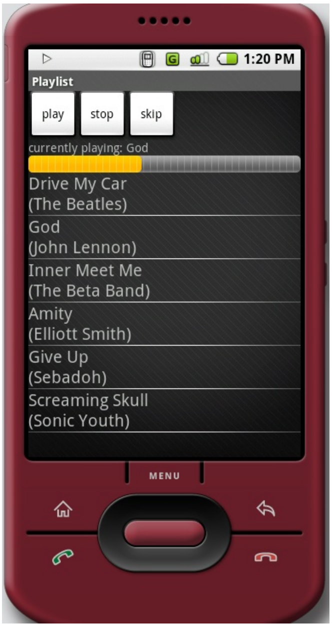
10.2 Textual Interfaces

We begin our discussion with simple textual interfaces designed for playlist generation and collection browsing.

10.2.1 Playlist Generation using End Points

As discussed before, a map of music allows the creation of smooth playlists by following a trajectory. The simplest case of such a trajectory is a straight line, which can easily be defined by selecting the two end points. We have implemented this idea using a textual interface that allows to select a start and an end song. The device then automatically produces a playlist of desired size or duration that contains only songs in between the two end points and thus exhibits a gradual transition in style.

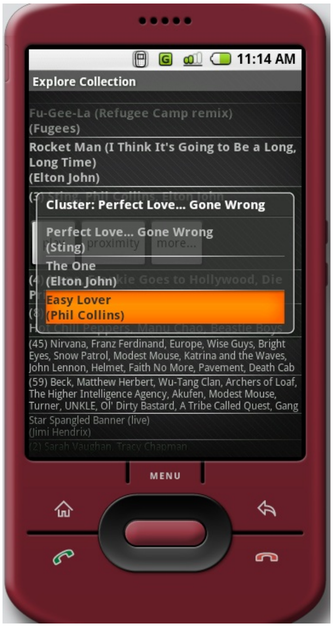
Further constraints (such as no artist repetitions) can be defined. Playlists are generated by a simple greedy algorithm. The line segment is divided uniformly according to the requested number of songs. Then to each of the resulting points the closest song not contradicting any constraints is selected. If the length of a playlist is defined by duration rather than the number of songs, the requested number of songs is derived from the average song length in the collection. Moreover, on each insertion of a song, the subdivision of the remaining line segment is recalculated (based on the remaining time), such that the final playlist typically well approximates the desired duration. An example playlist is shown in Figure 10.1(a). It can be seen that the playlist describes a gradual genre transition between the start point (The Beatles' "Drive my car") and the end point (Sonic Youth's "Screaming Skull").



currently playing: God

Drive My Car
(The Beatles)
God
(John Lennon)
Inner Meet Me
(The Beta Band)
Amity
(Elliott Smith)
Give Up
(Sebadoh)
Screaming Skull
(Sonic Youth)

(a) A playlist, created by the trajectory based method.



Fu-Gee-La (Refugee Camp remix)
(Fugees)
Rocket Man (I Think It's Going to Be a Long, Long Time)
(Elton John)
(3) Sting, Phil Collins, Elton John

Cluster: Perfect Love... Gone Wrong
Perfect Love... Gone Wrong
(Sting)
The One
(Elton John)
Easy Lover
(Phil Collins)

(4) ... Frankie Goes to Hollywood, Die Pr...
(8) ...
H... ot Chili Peppers, Manu Chao, Beastie Boys
(45) Nirvana, Franz Ferdinand, Europe, Wise Guys, Bright Eyes, Snow Patrol, Modest Mouse, Katrina and the Waves, John Lennon, Helmet, Faith No More, Pavement, Death Cab
(59) Beck, Matthew Herbert, Wu-Tang Clan, Archers of Loaf, The Higher Intelligence Agency, Akufen, Modest Mouse, Turner, UNKLE, Ol' Dirty Bastard, A Tribe Called Quest, Gang
Star Spangled Banner (live)
(Jimi Hendrix)
(2) Sarah Vaughan, Tracy Chapman

(b) Clicking on a group in a cluster-based interface.

Figure 10.1: Screenshots of the textual interfaces.

10.2.2 Textual Navigation through Music Collections

The Euclidean representation offers the advantages of similarity-based search. As opposed to keyword-based search, it allows to retrieve items, whose titles the user might not know, based on their similarity to known items, from which the user starts to navigate. Such a "proximity based" search is particularly important in a context where music collections are becoming large and dynamic.

In the following, we describe an interface that is based on simple textual lists, and that facilitates an intuitive similarity aware navigation. We thereby assume that the user owns a big music collection that contains many items. In particular, this number of items is much larger than the number of items that can be displayed in form of a list on the device's screen. An interface that facilitates an efficient navigation of the entire collection, needs to meet two basic requirements:

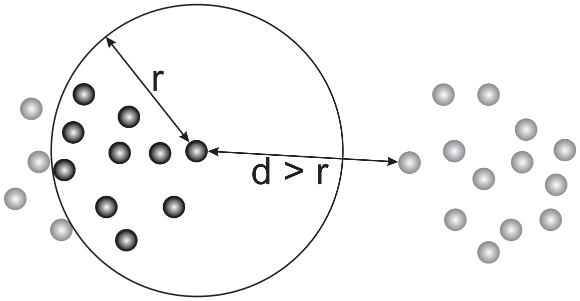


Figure 10.2: Getting stuck in a cluster.

- *Reachability*: The entire collection should be reachable from any given starting point.

- *Searchability*: Users should be able to quickly find what they are looking for. In particular, short paths should exist between any two songs. Moreover, the users should be able to detect these short paths, i.e. at any stage they should be able to select an item that brings them considerably closer to the goal.

An intuitive approach to explore a collection is to select a starting point (or *root*) and to present the proximity of this point in the list. Clicking on an item in the list allows to change the root and display this new root's proximity.

A naive solution might always put the k closest neighbors of the root song in the *displayed-list*. Unfortunately, this approach has several drawbacks. First, it might take a considerable number of clicks to move from one end of the map to the other. Second, if the tracks are not uniformly distributed in space, but form clusters, there is a risk of getting stuck in a cluster, and thus to violate the *reachability* requirement. This problem arises, if the distance to the next cluster is larger than the radius required to cover k tracks within the given cluster. This problem is illustrated in Figure 10.2.

We will next discuss two solutions to the problem of list-based navigation: *small-world navigation* and *neighborhood clustering*.

Small-World Navigation

Stanley Milgram's "Six-Degree-of-Separation" experiment did not only reveal that people are interlinked by astonishingly short acquaintance chains, but

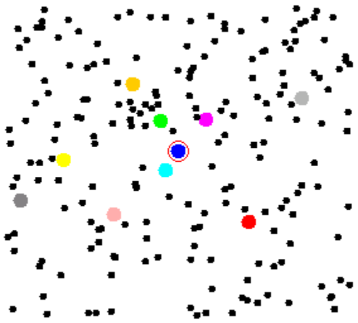


Figure 10.3: Small-world links: Nodes close to the root are more probable to be included in the list than nodes far away.

also that these short paths can efficiently be discovered by humans disposing of local knowledge only. Jon Kleinberg has later recognized that a specific edge length distribution ($1/r^d$, where d is the dimensionality of the underlying space) is required to achieve this navigability property [56]. He has shown that augmenting a grid (or the higher dimensional equivalent) by a single random outgoing link per node is enough to ensure polylogarithmic path lengths between any pair of nodes. Moreover, he has shown that these short paths can be detected using local knowledge only.

We take advantage of these insights by artificially overlaying our data with such an edge length distribution. Whenever a user selects a track, k random tracks are selected according to Kleinberg's distance distribution. These tracks are then sorted according to their distance from the root song, and the resulting list is presented to the user. Due to the specific distance distribution, the first items in the list are close to the root track, whereas the last items build the entry points to discover new areas farther away. Figure 10.3 illustrates the distribution of the listed tracks in a 2D sample space. The list is re-populated every time the user selects a root song. As a consequence, every track has a certain probability to appear, which ensures that any song can eventually be reached. Moreover, if a user is not happy with the "neighbors" offered in the list, he can simply re-select the same root and hope for a better list.

While this method might fail to quickly discover a particular song (as it might get missed by the random process several times), it provides an elegant light-weight solution to quickly get an overview of a music collection.

Neighborhood Clustering

Another approach to represent a collection in a navigable list is to make the list's items entire groups of songs, rather than single songs. In particular, the space can be recursively clustered according to the root song's proximity. Thereby, areas close to the root song are subdivided in a fine grained manner, resulting in small clusters. Distant areas, on the other hand, form large clusters. Each of the resulting clusters is then displayed as a single list-item in the interface, as illustrated in Figure 10.1(b).

An algorithm that creates such a recursive clustering will be presented in the context of visual browsing in Section 10.3.1.

10.3 Visual Browsing

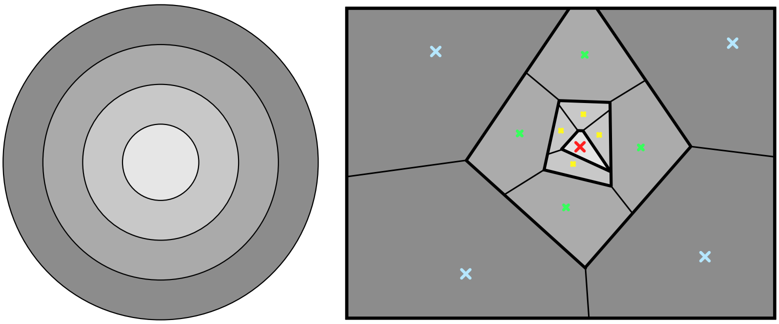
The taste of a single user is typically restricted to a few styles of music. Thus, a smart visual interface should group similar tracks and provide a facility to quickly guide the users to their favorite regions within the underlying space. Similarly to the reachability and searchability requirements for list-based navigation, we identify the following requirements for a visual music exploration scheme:

- *Locality*: When browsing in a specific region of music, the local proximity is interesting and should thus be in the user's focus.
- *Globality*: Simultaneously to the *local* information, more distant tracks should be visualized, making it simple to cross over different regions in the collection.
- *Orientation*: Moving from one region to another should be transparent and predictable to the user.

For typical 2-dimensional settings there exist several well known concepts to satisfy these criteria, such as fish-eye views, or a combination of zooming with satellite views. As indicated earlier, however, the space of music cannot directly be mapped into a low-dimensional representation. Thus, the mapping into the display space has to be dependent on the current area of focus. We will next introduce the *lens* metaphor that fulfills the outlined requirements and is applicable to the small screen of mobile devices.

10.3.1 The Lens View

The *lens* metaphor is directly derived from the (2-dimensional) fish-eye concept. The idea is to show the most detailed view in the center of the screen and to blur out more and more details with increasing distance from the



(a) Lens metaphor (b) Recursive partitioning of a 2D space

Figure 10.4: Lens view

area of focus. The distance from the center is schematically illustrated by means of rings, which indicate the lens (see Figure 10.4(a)). A user can define the area of focus by selecting a center song. In the innermost circle, few, highly related songs are then displayed. The outer circles contain clusters of songs, rather than single songs. With increasing distance from the center these clusters grow bigger and represent more distant regions of the space. This idea is illustrated in Figure 10.6(a). The uniformly colored points in the innermost circle denote songs similar to the center song (star), whereas the colorful symbols in the outer circles represent clusters of songs.

To realize this lens view, i.e. increasing cluster sizes with increasing distance from the center, we apply a technique which we refer to as *recursive k-means*. The algorithm, as the name indicates, clusters the space in a recursive manner, as schematically depicted in Figure 10.4(b) for a two dimensional space.

First, the entire (10-dimensional) space is clustered into a fixed number (k, 5 in our case) of clusters. For clustering, a modified version of the k-means algorithm is used, which fixes the centroid of one cluster to the center song. This results in $k-1$ outer and one center cluster. In the consecutive steps, the same procedure is applied to the songs residing in the center cluster. This process is repeated until either the center cluster contains less than q points, or until a maximum number of recursion levels is reached. The last condition prevents the generation of too many levels, which could not be visualized on small screens.

The initial cluster centroids are chosen at random. Hence, the space partition is not deterministic, and the result presented to the user varies each time the algorithm is run. This is not a flaw, but rather a desired behavior, as it allows to view the same area of focus in slightly different variations.[3]

Recursive k-means defines the content of the clusters to be used in the lens view. For visualization, the obtained positions (i.e. cluster centroids) in the high dimensional space have to be embedded into the 2-dimensional visualization plane. This mapping is done by a special spring embedder tailored to the needs of the lens view.

Similar to the hierarchical clustering, the visualization is also done in multiple steps. First, the songs belonging to the center cluster are embedded. Thereby the algorithm is restricted to only choose positions within the innermost circle. The remaining clusters are then successively embedded within the outer rings. Thereby each ring corresponds to one k-means recursion level. That is, the position of clusters stemming from a given recursion level are restricted to lie within one particular ring. The embedding process is directed from the center towards the border. Already embedded points of the inner rings are kept fixed but contribute to the force that acts on the points that are being embedded.

To guarantee a consistent view, the initial positions of the points are set corresponding to the first two dimensions of the high-dimensional space. Thus the outermost clusters are always placed at similar positions (e.g. the area containing mostly electronic music is always placed in, say, the upper right corner). Observe that this strategy also reduces the number of iterations (as compared to random positions), as the initial positions are likely to be close to the final positions.

10.3.2 The Cake Metaphor

In Section 9.2.2 we have seen that the LMDS space exhibits a fairly good clustering in terms of genres. We have thus augmented the obtained cluster centroids with genre information, to make browsing more intuitive. The center of mass of positions of songs with known genre information (acquired from the Allmusic Guide) defines a centroid for each genre. Measuring the distances to these genre centroids allows to define a genre relationship to each point in the 10-dimensional space.

This information is visualized using the *cake* metaphor, which consists of an inner circle and an outer ring. The outer ring is subdivided into segments – or *cake* slices – that represent the inverse squared distances from the different genre centroids. The inner circle represents the nearest cluster. The hue (i.e. color) of the inner circle corresponds to the hue of the nearest cluster, whereas its saturation indicates, how clear the given position could be assigned to the corresponding genre. A cluster centroid which has almost the same distance

[3]Observe that a deterministic behavior could easily be achieved by the use of random seeds.

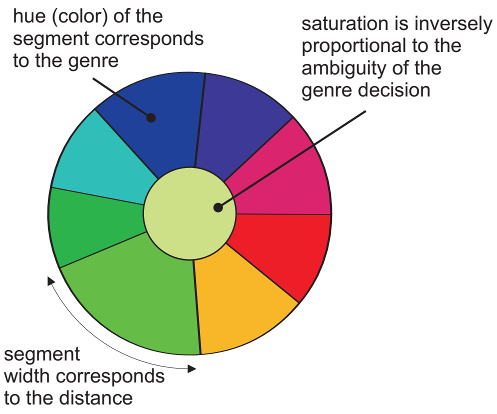

hue (color) of the segment corresponds to the genre
saturation is inversely proportional to the ambiguity of the genre decision
segment width corresponds to the distance

Figure 10.5: Cake metaphor

to all genre centroids, for example, is represented by a very pale color. An example of a cake diagram is depicted in Figure 10.5.

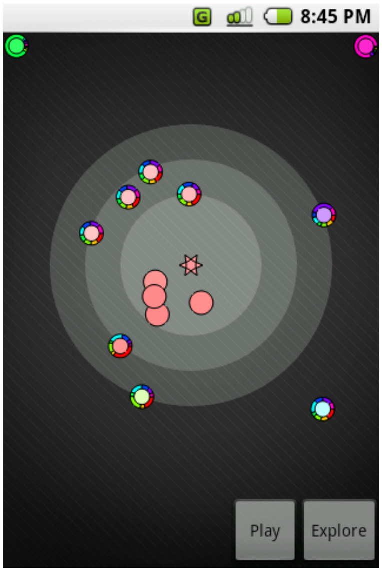
Observe that both, the lens as well as the cake metaphor are not restricted for use in music collections in conjunction with genres. Rather the lens metaphor can be used in conjunction with any high-dimensional space and, similarly, the cake metaphor applies to any meaningful anchor points, be it for music collections (where, e.g., moods could be used), or any other high-dimensional data spaces.

10.3.3 The Final Interface

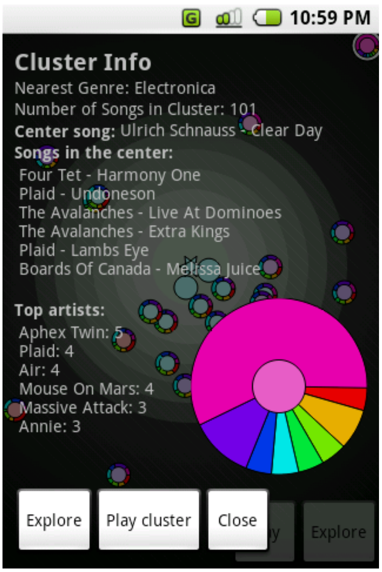
The final interface combines the cake with the lens metaphor. The cake metaphor is used to visualize the clusters in the outer rings. Each song in the center area is represented by a simple circle, colored analogously to the inner circle of a cake diagram. The resulting interface is illustrated in Figure 10.6.

The different elements (songs and clusters) in this interface react to touch events. Selecting a single song in the inner circle allows to either play it, or to make it the new center song. Selecting a cluster opens a dialog (see Figure 10.6(b)) that presents the cluster's detailed content. Moreover, the dialog contains a button to play the contained songs, as well as a button that allows to move to the cluster's centroid (meaning the song closest to the cluster's centroid becomes the new center song).

The primary goal of exploring a music collection is to later listen to it. We have thus incorporated a playlist creation mechanism. To create a playlist a



(a) The lens metaphor with visualized clusters



Cluster Info
Nearest Genre: Electronica
Number of Songs in Cluster: 101
Center song: Ulrich Schnauss - Clear Day
Songs in the center:
Four Tet - Harmony One
Plaid - Undoneson
The Avalanches - Live At Dominoes
The Avalanches - Extra Kings
Plaid - Lambs Eye
Boards Of Canada - Melissa Juice

Top artists:
Aphex Twin: 5
Plaid: 4
Air: 4
Mouse On Mars: 4
Massive Attack: 3
Annie: 3

(b) Detail information for a selected cluster

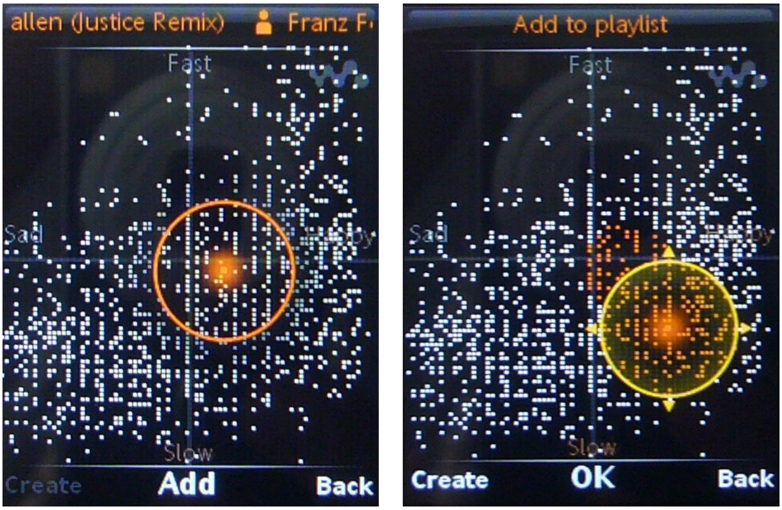
Figure 10.6: Our graphical user interface for visual browsing.

user can add each center song to a set of *seed* songs. Once enough seed songs are selected, a playlist consisting of the chosen seed songs as well as of songs randomly sampled from each seed song's neighborhood can be generated. A user settable parameter allows to control the size of these neighborhoods and thus the diversity of the resulting playlist.

10.3.4 Evaluation

For evaluation we have conducted a preliminary user study with 9 participants. The study mainly compares our interface with SensMe, as shipped with Sony Ericsson phones. To the best of our knowledge SensMe is currently the only commercial visual interface for music exploration on mobile devices.

SensMe is based on audio-analysis and classifies the songs according to the two properties tempo (slow vs. fast) and mood (sad vs. happy). The songs are arranged on the screen along these two axes, as illustrated in Figure 10.7. To create playlists, circular areas of adjustable size can be selected. The final playlist then contains the union of all the songs within the selected areas.



(a) Browsing (b) Adding areas to a playlist

Figure 10.7: SensMe: A commercial visual browsing interface shipped with Sony Ericsson devices.

The main part of the user experiment consisted in the creation of a playlist (20 songs) with both interfaces. In addition, the participants had to fill in a questionnaire. For both, SensMe as well as our interface, the participants were given the same collection, which they were not familiar with. The collection contained approximately 1400 files (7.5GB) and covered a broad range of music. For each interface, the participants were given 5 minutes to create a playlist that matched their taste and mood as closely as possible. Afterward they had to listen through the playlists and rate each song on a scale from 0 (worst) to 10 (best). The resulting average ratings were 6.3 (our interface) and 5.5 (SensMe).

In addition, we prepared a small questionnaire the participants had to fill in. If not indicated otherwise, the answers were measured on a scale ranging from 1 (worst) to 5 (best). The major findings can be summarized as follows:

- *Playlist (overall)*: We were not only interested in the rating of individual songs, but also asked the participants to judge the overall impression of the obtained playlist. The result is in line with the song ratings (3.33 (our interface) vs. 2.44 (SensMe)) and suggests that the presented interface is in fact better suited for playlist generation.

- *Diversity*: People were asked to judge the playlist diversity on a scale from 1 (too diverse) to 5 (not diverse enough), with 3 as a neutral value (just right). While our algorithm is not quite diverse enough (3.44), the playlists generated by SensMe are too diverse (2.44). The result suggests that both algorithms perform similar in this respect with slight advantages for our algorithm.[4]

- *Usability*: We wanted to know whether the interface is intuitive to use. As expected, the simple 2-dimensional interface of SensMe outperformed our exploration scheme for a high dimensional space. The ratings were 4.67 (SensMe) versus 3.67 (our interface).

- *Underlying space*: We asked our participants whether, in their opinion, similar songs well group together in the respective system. In this question our approach (4.00) clearly outperforms SensMe (2.44). A possible reason for this result is that the two dimensional representation of SensMe is not capable to adequately reproduce the underlying similarity. It might thus indicate that it is worth to operate in higher dimensional spaces at the expense of a less intuitive interface. Another explanation is that the user-behavior driven similarity measure underlying our interface provides better results than the audio-analysis based methods of SensMe.

- *Overall*: We wanted to know whether our users would use the system again. Thereby, only the answers *yes* and *no* were allowed. Again, the result favors the interface presented here (67% yes) over SensMe (44% yes), which is in line with the playlist ratings discussed earlier.

Finally, we wanted to know how useful the cake metaphor is. The results show that the most valuable part is the center circle color (3.22). Cake slices (2.67) and the center circle saturation (2.55), however, have also been used for navigation.

10.4 Acoustic Exploration

As music is primarily perceived by the sense of hearing, acoustic navigation through the space of music also seems to be a natural approach. Similar as in the presented visual exploration scheme, the goal of our acoustic interface is to quickly guide users to music of their taste.

[4]Observe that our algorithm provides a parameter to control diversity. We can thus expect that people that regularly use the system are able to produce better playlists, as they get a better feeling for the right parameter setting.

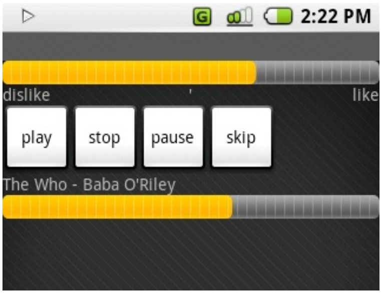


Figure 10.8: Acoustic exploration interface: Note the rating bar at the top.

As opposed to the visual situation, it is impossible to acoustically provide an instant global impression of the space, as we are not able to listen to many tunes in parallel. As pointed out by Tzanetakis and Cook [118], 8 simultaneous tunes form an upper limit. An acoustic exploration scheme thus has to rely mainly on local information. Moreover, people's notion of orientation or direction in acoustic space is much more fuzzy than in geometric space. As a consequence, the control of the exploration process should shift from the user towards the device.

The goal of our interface is to dynamically generate a sequence of songs (a playlist) that fits the user's taste. This is achieved by constantly appending new items to the sequence dependent on the user's (implicit or explicit) feedback about the preceding songs. Thereby, the algorithm learns the user's taste and, ideally, selects ever better items.

Our primary interface uses explicit user feedback defined on a continuous rating scale ranging from 0 to 1 (see Figure 10.8). For scenarios that do not allow for explicit feedback, we offer the possibility to switch to a binary scheme that assumes a song was disliked if it was skipped, and liked otherwise (analogously as proposed in [95]). A continuous rating scale, however, allows to more precisely estimate and react to the user's needs. If only skipping behavior is considered, this interface looks much like an ordinary shuffling mode, such as available in most music players. However, by reacting to the user feedback, it is able to smartly select songs matching the user's taste. In the context of *museek* we thus refer to this kind of interface (or play mode) as *smart shuffling.*

Before we come to our exploration algorithm in more detail, let us briefly review its major requirements:

- *Taste*: Clearly, the exploration algorithm should visit areas and songs the user likes.

- *Diversity*: People tend to get bored when listening to several very similar (e.g., same artist) songs in a row. The exploration scheme should thus also provide sufficient diversity.

- *Adaptability*: The algorithm should be able to adapt to the changing mood of a person. That is, if, after some time of listening to rock music, the listener starts feeling more like jazz, the algorithm should be able to quickly adapt to this changed condition.

- *Exploration*: As the name suggests, the exploration scheme should be able to *explore* a collection. That is, it should not just stick to songs the user listens to frequently, but should be able to find songs the user might even not have been aware of.

10.4.1 Acoustic Exploration Algorithm

The basic idea of our exploration scheme is that on completion of a song, the algorithm looks at the ratings acquired so far and then selects the next song in an intelligent manner. A naive algorithm might simply select the song most similar to the currently best rated song, thereby only considering songs that have not yet been played. This method, however, would most likely violate the *diversity* requirement. We thus follow another strategy, which is based on the Voronoi tessellation.

We start by outlining a simplified version of the algorithm, which works as follows: The previously rated songs are used as the input of a Voronoi tessellation, which is applied to the entire collection. As a result, each previously rated song becomes the *generating point* of one Voronoi cell. We assume that ratings above 0.5 (on a scale from 0 to 1) indicate that the user likes a song, whereas lower ratings are seen as a sign of dissatisfaction. As a consequence, each Voronoi cell (and thus also all the contained songs) can be seen as either *good* or *bad*, dependent on the rating of its *generating point*. The Voronoi tessellation is re-executed on each song completion. After tessellation the exploration algorithm selects a not yet played song from within a *good* Voronoi cell. This strategy is schematically illustrated in Figure 10.9. The lighter areas, which indicate *good* cells, roughly approximate the user's region of interest (shaded).

The entire process is started by selecting a song uniformly at random from the entire collection. If this song is rated *good*, everything continues as outlined before. A rating below 0.5, however, would lead to a single Voronoi cell that is considered *bad* and covers the entire collection (resulting in no playable songs). Whenever no positively rated song is available, we therefore add a *floating centroid* to the system, the position of which is determined

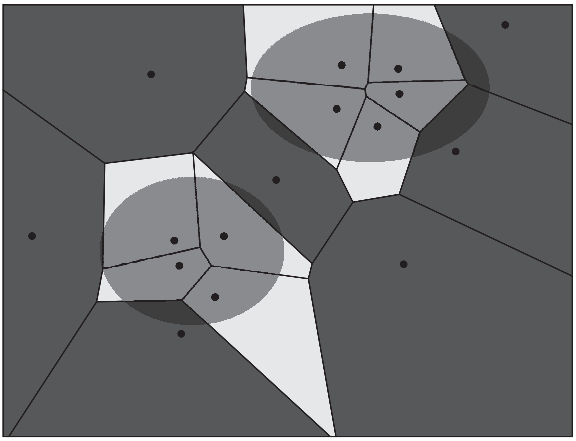


Figure 10.9: Acoustic browsing (simplified): The entire collection is decomposed by means of Voronoi tessellation. The generating points for the Voronoi cells are given by the rated songs. Dependent on the rating each cell is either considered *good* (light gray) or *bad* (dark gray). The shaded areas indicate the user's region of interest. Note that in the application a 10 dimensional, rather than a 2 dimensional space, as illustrated here, is used.

by the modified version of the k-means algorithm described in the visual exploration part (keeping all previously rated positions fixed). This procedure is repeated until the first positively rated song is found. Afterwards, the floating centroid is removed.

Observe that this simple algorithm exhibits various weaknesses. First, it does not take advantage of a continuous rating scale. Moreover, it is not able to satisfy the *adaptability* requirement, since regions once marked *bad* will never be visited again. Finally, the *good* Voronoi cells only provide a rough approximation of the region of interest. In particular the border areas of these cells are likely to disagree with the user's taste. To overcome these shortcomings we improve the outlined scheme in several points:

- *Weighting*: Dependent on the rating of the corresponding generating point, a weight is assigned to each Voronoi cell. The more this rating deviates from 0.5 (in either positive or negative direction), the higher the weight. When selecting the next song to play, the algorithm considers these weights by selecting a (good) song with a probability proportional to the weight of its enclosing Voronoi cell.

- *Aging*: Weights are not static. Rather, on each song completion, all weights are reduced by some constant value. If the weight of a generating point falls below a certain threshold t, the corresponding point is removed. Due to the aging technique, old ratings eventually become obsolete. Thus, the algorithm can well react to changes in the user's mood.

- *Centering*: Songs close to a positively rated generating point lie within the user's region of interest with higher probability than songs at the border of the corresponding Voronoi cell. Thus, such songs are selected with higher probability by the algorithm. For simplicity, we currently use a fixed probability distribution. However, making this distribution adjustable by means of a parameter might allow the user to control the sequence's diversity.

- *Escaping*: At some point, the algorithm might start playing songs from within a certain narrow area only, as preliminary experiments have shown. The missing diversity is then likely to become manifest in form of repeated low ratings. We thus allow the algorithm to break out of such gridlocked situations by selecting a completely random song with a certain probability. This probability is adaptive within a given upper and lower bound. It decreases with each positive, and increases with each negative rating.

These tweaks have shown to improve the quality of the resulting sequences. However, our first experiments have also revealed some performance issues. After all, a user is not willing to wait for several seconds until the next song is being played. As we only want to select a single song in the end, it is obviously an unnecessary overhead to classify each song as either *good* or *bad* in each round (i.e. after each song completion). Instead, we move part of the random selection process to the beginning of a round by first selecting a random sample of the not yet played songs.

10.4.2 Evaluation

For evaluation we have compared the algorithm to two alternative approaches:

- *Shuffling*: Most music players offer the possibility to listen to a collection in a purely random fashion. Moreover, 7 of our 9 participants said that they at least sometimes do listen to music completely randomly.

- *Pampalk*: Pampalk et al. [95] have proposed an algorithm that dynamically creates playlists based on the user's skipping behavior. Although

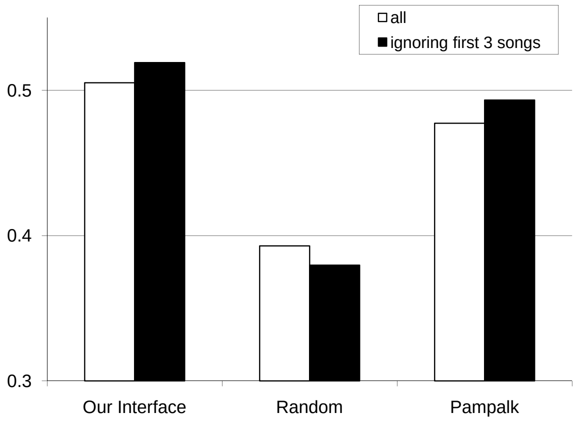


Figure 10.10: Comparison of the three different acoustic exploration methods. The light bars indicate the average rating over all songs. To demonstrate the learning effect, the dark bars ignore the ratings of the first 3 songs in each sequence.

the method was originally proposed for use in conjunction with an audio-feature space, it can be applied to our space, too. The algorithm works as follows: Each song for which the nearest *good* song is closer than the nearest *bad* song is added to a set S. If S is non-empty, its element with smallest distance to the nearest *good* song is selected. Otherwise, the song with the lowest d_g/d_b ratio is played, where d_g denotes the distance to the nearest *good*, and d_b the distance to the nearest *bad* song. The algorithm does not make use of rated feedback, but uses a binary scheme solely based on skipping behavior.

For comparison, all participants had to create a sequence of 20 songs with each of the three methods. In all cases, the start song was chosen randomly. Moreover, we let the users uninformed about the internals of the different algorithms to avoid biased results. They were, however, told that the algorithms were supposed to find their taste (which is not exactly true for random shuffling). We used the interface with explicit song ratings (recall Figure 10.8) for all experiments. The acquired ratings were not only used to feed the algorithms, but also to later assess the qualities of the different playlists.

The results are summarized in Figure 10.10. The figure depicts the overall average song rating for each algorithm (light bars), as well as the average

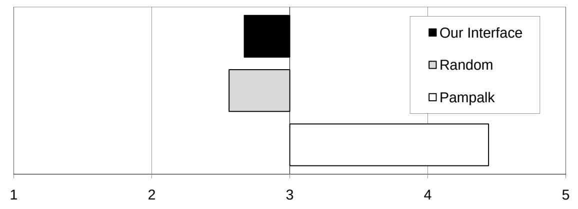


Figure 10.11: Diversity: Average values for the diversity ratings of the participants (1: too diverse, 3: just right, 5: not diverse enough).

rating after ignoring the first three songs of each sequence (dark bars). For both algorithms that are supposed to adapt to the users taste, the dark bars are higher, indicating that the algorithm in fact improved over time. In the random case, by contrast, the rating got worse, which might be explained by the unsatisfied user expectation. The figure further reveals that the adaptive algorithms by far outperform random shuffling. Moreover, we can see that only the algorithm presented here reaches a positive average rating (i.e., above 0.5).

As stated earlier, we were not only interested in the overall rating of the resulting sequences, but also in the algorithm's capability to explore a collection. We have assessed this property by means of participant questioning.

Using a questionnaire, we had the participants judging their satisfaction concerning the sequences' diversity on a scale from 1 to 5 (1: too diverse, 3: just right, 5: not diverse enough). The results are summarized in Figure 10.11. As expected, the diversity of the random shuffling algorithm was considered too high (average: 2.56, min: 1, max: 4). The diversity of the algorithm of Pampalk et al., on the other hand, was considered to low by all participants (average: 4.44, min: 4, max: 5). The best score (i.e. closest to 3) was reached by our algorithm (average: 2.67, min: 1, max: 5), which scores slightly better than random shuffling, and clearly outperforms the approach of Pampalk et al.. The fact that for one single algorithm the entire rating scale has been used suggests that diversity is a very subjective measure.

It is important to note that Pampalk's algorithm possibly provides more diversity if applied to audio-feature spaces, for which the algorithm was originally designed. Due to the strong dependence of diversity on the user, and, possibly, the underlying space, we believe that an algorithm should be adaptive in this respect (either by means of a user settable parameter, or by means of self-regulation). As mentioned before, we could realize this idea by incorporating a parameter that controls the *centering* effect.

10.4.3 Community-Aware Interactive DJ

Music does not only coin the lives of individuals, but also plays an important role when people socialize. In the following we will describe how the acoustic exploration scheme can seamlessly be applied in such situations. We assume a scenario, such as, say, a private party.[5] Clearly, the guests should be entertained with music of their taste throughout the evening. Hiring a professional DJ most likely exceeds the host's budget. The probability that one of the guests is able to grasp the desires of the audience, and willing to act as a DJ is rather low. The alternatives are to connect somebody's iPod in random shuffling mode, to select an entire album from time to time, or to compel somebody to act as DJ. All of these options are likely to result in a (possibly undesired) controversy about the selected songs.

In the following we want to briefly sketch an idea that could overcome these issues. The only input required by the acoustic exploration algorithm is feedback about the played songs. It is indifferent, however, to whether the ratings stem from an individual, or from a group of guests. Hence, if we are able to acquire some sort of rating from the audience, the exploration scheme is able to create a sequence of songs that matches the majority's taste. Clearly, asking for explicit feedback from each guest is not appropriate. However, state-of-the-art mobile phones are more and more getting equipped with motion sensors, which can provide our algorithm with valuable information. Surely, people tend to dance more to music they like. By means of motion sensors we can estimate the fraction of people dancing. This data can then be converted into a rating which serves as the input for acoustic exploration.

We have conducted preliminary experiments that show that dancers can quite well be distinguished from non-dancers by comparing the audio signal's beat to the motion patterns. Further research, however, is required to investigate how well such a system is able to adapt to the crowd's taste in a real-world setting.

The guests could profit from such a system in two ways. First, the played music better matches their taste. Second, each client device could maintain a list of the owner's favorite songs (i.e. the songs with highest dancing activity). Consulting this list after the party allows to learn about new music previously unknown to the user.

10.5 The Interfaces Revisited

The interfaces presented so far demonstrate the practical usability of our (LMDS) music map. However, they only act as a proof of concept. In par-

[5]The same idea also works in other settings, such as in public places, pubs, etc.

ticular, they have only been used in small scale user studies and were never made available to a large listening community. Both, the conducted experiments, as well as informal discussions with various people have uncovered relevant issues that should be improved before targeting a large user base. The most important lessons learned were:

- Users tend to stick to concepts they are already familiar with.
- The interfaces should be as simple and intuitive as possible, even if this comes at the cost of accuracy.
- Users are not interested in what goes on behind the scenes. They only judge the result.
- The traditional interfaces (such as hierarchies of artist, album and track lists) are indispensable in an application that should be accepted by the user.

Based on these findings, we have implemented *museek*, a music player designed for the Android platform. In particular, we have taken care that the visualization interface becomes more intuitive, and we have improved upon the acoustic exploration algorithm. For this purpose we introduce a collection dependent 2-dimensional representation that facilitates the navigation as in a traditional map interface. Moreover, we have found that smart shuffling becomes more reactive when operating in 2 rather than 10 (or more) dimensions, as the relevant regions can be narrowed down with fewer decisions.

In the next section we will describe the application in more detail, and we will present a study based on usage logs that shows that the novel interfaces are appreciated by the community.

10.6 A Comprehensive Music Player

In an attempt to address the users' needs (as outlined in Section 10.1) and to demonstrate the usefulness of a music map, we have developed *museek*, a music player for Android smart phones that provides similarity based functionality. Our player incorporates the following features to access and discover music: (1) Traditional alphabetic lists (song, album, artist, and genre) to browse for music, (2) a full text search option to search in the title and artist fields, (3) a tag-cloud to select music by social tags, (4) a music map to visualize a collection, (5) a play mode that plays songs similar to the previous one, and (6) a play mode that avoids inappropriate regions by considering skipping behavior.

Features (3) to (6) rely on the PLSA map described in Chapter 9. The coordinate information is made available to museek in an initial import process. Thereby, the coordinates are fetched for each song in the user's collection according to the artist and title tags. In case a song is not available in our database, the corresponding artist coordinates are returned (recall that we have calculated artist coordinates for more than 1M artists). Once this import step is completed, similarity queries can be performed without the need for an Internet connection. Measuring similarity between two songs reduces to calculating the distance between the corresponding coordinates.

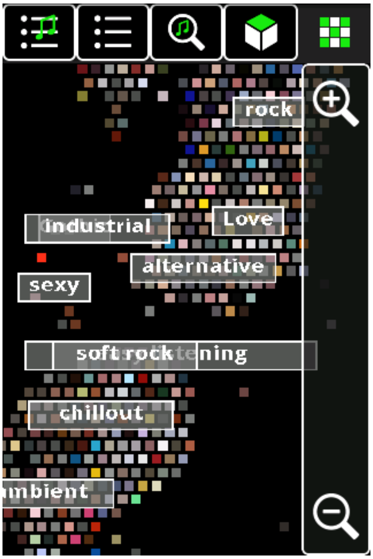
For visualization, our high dimensional space is transformed into a collection dependent 2D map on the user's device using Principal Component Analysis (PCA). The PCA was not applied to all songs, but only to the center of mass of the most relevant tags in the collection. Although, in Chapter 9, we have stated that a 2 dimensional embedding can not accurately represent general music similarity this might be possible for the collection of a user, as it is likely to contain only a small subset of the overall music diversity.

The traditional search options, i.e. features (1) and (2), do not need any further explanations. Rather, we quickly want to sketch the most important properties of the other interfaces. Our music map combines the strengths of Apple's Cover Flow and Sony Ericsson's SensMe interfaces. The popularity of Cover Flow shows that album art is a good visual hint to recognize music and also stresses the user's desire for visually attractive ways to browse a music library. SensMe, on the other hand, provides a neat way to explore a collection and to quickly create appealing playlists by selecting regions from a map. We have thus implemented an intuitively navigable map that uses album covers as visual aids.

At low zoom levels (see Figure 10.12(a)) the map resembles the point cloud of SensMe. Tags help the user to keep an overview. When zooming in, the points become recognizable as album images (see Figure 10.12(b)). From this view the user can either select an album to be played or discover other, similar albums by browsing through the covers. The map can be navigated by moving the finger on the touchscreen, much like this is done in traditional map interfaces. On Android devices that lack multi-touch support, a zoom bar on the right allows to zoom in and out smoothly by wiping the finger over the bar. Moreover, a touch gesture allows to select a region from the map to create a playlist similarly as in SensMe.

Besides the described 2D mode, the map also comes in a 3D flavor that focuses on offering an appealing browsing experience. The 3D view uses the same underlying 2D space but arranges the album covers in 3D (see Figure 10.13(a)). Wiping the finger up and down allows to move backward and forward respectively.

We have also used the music similarity map to offer two novel play modes,



(a) The map from far away

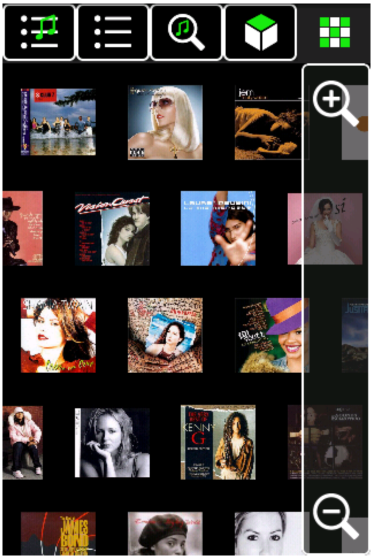
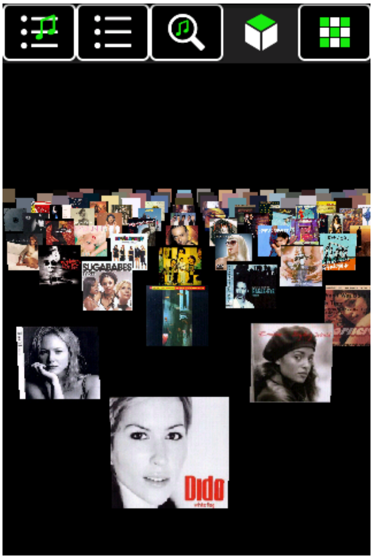
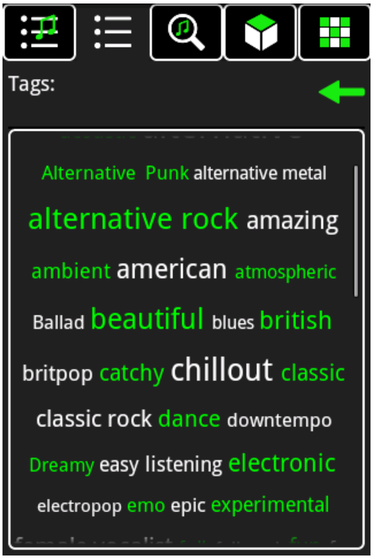

(b) A close look at the similarity map.

Figure 10.12: The 2D music similarity map

a *similar song mode* and a *smart shuffling mode*. The idea of the similar song mode is to extend an existing playlist with similar songs. This means a user can select a start song he/she likes and the application will automatically add songs that are similar to the playlist. This allows the generation of smooth playlists by choosing a single seed song. To avoid that the *similar song mode* plays songs from just one artist, the user can specify that an artist may not re-occur for a certain number of subsequent songs.

The smart shuffle mode selects songs from the entire collection and thereby intelligently avoids music styles of songs the user has previously skipped. The implementation basically corresponds to the acoustic browsing method introduced in Section 10.4. That is, the idea is to subdivide the map into good and bad regions. However, we operate on a 2 rather than a 10-dimensional space. Moreover, we have decided to fully adhere to the implicit interaction metaphor [105]. That is, we have omitted the rating bar, and solely rely on skipping behavior to make the interface more user friendly, and controllable from the headphones' media buttons. A region is marked good if the corresponding songs were listened to the end, and bad, if the songs were skipped. Observe that more precise rating information could be acquired when taking additional cues into account, much as this has been


(a) The music similarity map in 3D


Tags:

Alternative Punk alternative metal alternative rock amazing ambient american atmospheric Ballad beautiful blues british britpop catchy chillout classic classic rock dance downtempo Dreamy easy listening electronic electropop emo epic experimental

(b) An auto-generated tag-cloud for a user's collection.

Figure 10.13: Screenshots of the 3D map and the tag-cloud.

proposed for web usage [3]. Such cues might for example be the time after which a song is skipped, or changes in the volume control. However, for transparency and simplicity reasons, we do currently not consider such additional cues.

Finally, we have seen that users often describe their needs in terms of genres or other descriptive information, such as mood. Thus we offer the possibility to choose songs by selecting a tag in a tag cloud (see Figure 10.13(b)). This tag cloud is individually generated for a user, displaying only tags that are relevant to the music collection on the device. As the tags in this cloud are freely generated by last.fm users, they do not only specify genres and sub genres but also moods and feelings. This facilitates a fine grained selection of the desired music. That is, if a user selects a tag, the song that exhibits the highest probability to generate this tag (recall Equation 9.7) is identified. Afterwards, songs in the corresponding neighborhood are played by *museek*.

The outlined functionality is integrated into our player in 5 tabs. The *Player Tab* contains the player controls, the playlist, as well as buttons to control the play mode (repeat, shuffling over playlist, shuffling over collection, similar songs, and smart shuffling). The *Lists Tab* allows to access traditional alphabetic lists (namely song, album, artist, and genre list), as well as a tag-cloud (recall Figure 10.13(b)) to select music. A full text search mode is provided by the *Search Tab*, and, finally, there are two screens for the 2D and the 3D map, respectively (recall Figures 10.12 and 10.13(a)).

10.6.1 Evaluation

We have published our application on the Android Market (the App Store for Android) where it was downloaded more than 40,000 times at the time of writing. At the first startup, we ask the user for permission to log (anonymous) usage data. We removed overly short log files, as we were interested in the usage of regular users (as opposed to those that only had a quick look at the application). The following statistics are based on the remaining 128 data logs, each of which documents the application usage for a period of 5 days.[6]

To get a rough impression about how the application is used, we measured the times the users spent in the different tabs. Not surprisingly, the Player Tab, mainly used to listen to music, is the most popular view – users spend about two thirds of the time in it. The remaining time can be seen as the time spent to search or browse for music and is distributed as follows: Lists Tab (including tag-cloud) 53%, Map Tabs 40%, and Search Tab 7%. The fact that when searching for music the users spent 40% of the time in the Map Tabs confirms the need for serendipitous browsing options and shows that this interface is well accepted.

Studies about user needs suggest that people would often select music based on descriptive information, such as genre or mood. We have found that 51% of our users have at least once selected music from our tag-cloud, and that 19% used this feature regularly (3 or more times). Interestingly, only about 40% of the selected tags correspond to some genre, the remaining 60% reflect some mood (e.g. "happy","catchy") or subjective opinion (e.g. "beautiful", "amazing"). These numbers underline that genres alone are not descriptive enough to satisfy the users' needs.

The music player offers five different play modes: Repeat all songs of a playlist, shuffle over a playlist, shuffle over the whole collection, smart shuffling, and the similar song mode. The first two play modes define only the order in which a given set of songs is played. By contrast, the three other modes generate playlists by themselves, i.e. upon completion of a song they automatically select a new song to be played. Thus, we can distinguish between *explicit selection* of tracks (from the traditional lists, the tag-cloud,

[6]The big discrepancy between the number of downloads, and the number of usage logs is due to the fact that we have conducted the usage study in the early days of the application, when the user base was much smaller.

	Explicit plays	Implicit plays
Meta data lists	16%	-
Similarity map	8%	-
Tag cloud	10%	-
Shuffle	-	2%
Smart shuffle	-	51%
Similar mode	-	13%
Sum	**34%**	**66%**

Table 10.1: A comparison between the origins of the played songs. The new player features are used often to generate playlists and select music.

the search module, or the maps) and *implicitly generated suggestions* (from one of the latter three play modes). Table 10.1 shows how the listened songs are distributed among these different song selection methods.

Interestingly, only about a third of the music was explicitly selected by the user. The other two thirds were selected implicitly by the player, using one of the mentioned play modes. Moreover, we can see that the traditional search options account for less than half of the explicit selections, the remaining selections either occurred from the tag-cloud or the map. Considering the implicitly selected songs, we see that the similarity aware modes enjoy a great user acceptance. Surprisingly, the collection shuffling mode, which is prevalent in state-of-the-art players, is barely used.[7]

Comparing these results to the studies shown in Section 10.1, it is no surprise that the traditional lists are the most popular means to explicitly select music. The usage numbers of the descriptive and visual browsing methods, however, are even higher than predicted. This might reflect the fact that people only become aware of certain retrieval techniques, once they are provided with them, but then understand the advantages. Moreover, the number might be slightly biased from the user selection process. That is, users that downloaded our application might be disproportionally open to explore new features.

The plain numbers might let room for speculations. However, most of the results are clear enough to conclude that the novel features were well accepted by the users. Moreover, public feedback in the Android Market underlines the usefulness of similarity aware play modes. Considering smart shuffling, for example, people wrote:

> "[...] Does a good job learning my tastes. [...]"

[7]As smart shuffling is the default mode in our player, the result is likely to be biased. However, the large differences clearly underline the advantages of similarity aware play modes and show that there is little incentive to use collection shuffling if a smart shuffling mode is present.

> “Great app, learns what I like.”

Other comments confirm the acceptance of the similar song mode:

> “[...] easy browse and make playlists. Auto play related music is very good. [...]”

> “[...] Love the ability to automatically play similar music. [...]”

10.6.2 Conclusion

museek is a music player for Android mobile phones that facilitates novel ways to browse and find music. Our usage data based evaluation shows that these new features are well accepted and add to the user’s music listening and browsing experience. All the incorporated features, namely the similarity aware play modes, the music map, as well as a personalized tag cloud were highly appreciated. This shows that explorative browsing and descriptive music selection methods are indeed important to satisfy the users’ needs. Traditional methods, however, have also shown to be relevant. Thus, we conclude that future mobile music players should, in addition to standard bibliographic selection and browsing functionality offer sophisticated retrieval interfaces.

Chapter 11

Concluding Remarks

Looking at books in the bookstore, buying tools in the supermarket, selecting articles to read in the newspaper, and renting videos from the rental service, all these actions do not only fulfill their primary purpose, but also carry a huge amount of implicit information – about ourselves, but also about the items in question. A long time ago, large supermarket chains started to exploit this information. According to a legend, Wal-Mart once put beer and diapers next to each other in the shelf, as they were often sold together. Before the Internet age, it was a privilege of large companies to possess such information. With the emergence of the Internet, and the Web 2.0 in particular, user actions are recorded on a tremendous scale. Moreover, a considerable amount of this information is (possibly in an anonymized fashion) publicly available, and thus accessible to research.

The second part of this thesis was devoted to the analysis of such user generated data, with a special focus on similarity. We were thereby interested in the similarity among items rather than users, and thus investigated the mass behavior, rather than the behavior of individuals. More precisely, we have studied the publication records of individual authors to derive a similarity measure between scientific conferences, and we have looked into the listening history of last.fm users to construct a music similarity space.

Our analysis of publication data has shown that the knowledge about who has published which paper at which conference facilitates the extraction of different relationships among publication venues. In particular, we have seen that the paper titles can be used to thematically relate conferences. Moreover, the knowledge about the publishing authors defines another measure that can be seen as a mixture between similarity with respect to quality and thematic scope. We have shown that by combining the purely thematic, and the mixture measure we can emphasize the quality aspect. This result confirms the advantages of looking at different layers of networks, as suggested in the first part of the thesis. To demonstrate the practical usefulness of the derived similarity information, we have proposed a novel conference rating method, and we have implemented *confsearch*, a search engine for scientific conferences, designed to support researchers looking for an appropriate place for publication.

Similarly to how the co-occurrence of two conferences in the publication list of an author indicates that these conferences are somehow related, the co-occurrence of two songs in the listening history of a user indicates that the two songs are somehow related. We have made use of this observation to derive a music similarity measure from last.fm data. In contrast to audio-based approaches, such usage data derived measures are known to better reflect perceived music similarity. However, co-occurrence based measures only define pairwise similarity which restricts the possibilities of user-interface designers. To overcome these issues, we have introduced the concept of a music map, which is basically a Euclidean space that reflects socially derived music similarity information. We have shown two methods to construct such a music map, one that solely relies on the co-occurrences of songs in the users' listening histories, and one that takes user generated social tags as an additional input. The first method is based on graph embedding, whereas the second approach builds upon on a probabilistic framework originally developed for textual information retrieval.

Based on our music similarity maps, we have implemented a variety of interfaces that have not previously been accessible to social music similarity measures. Examples include a trajectory based playlist generator, a smart-shuffling algorithm that avoids unwanted regions based on skipping behavior, and interfaces to visualize a collection on small screens. Some of these interfaces have been integrated into *museek*, a publicly available Android music player. The high user acceptance of the player in general, and the similarity aware features in particular, underlines that the described music similarity spaces are well suited for the use in real-world settings.

These examples show how the end-users can profit from data they have implicitly created while following completely different goals. Clearly, we could only demonstrate this phenomenon exemplarily, on a very small sample consisting of scientific conferences and music. However similar techniques can doubtlessly be applied in other domains, too.

The ever growing archives of collected usage data are often criticized for privacy reasons. These privacy concerns should definitely be taken serious. In particular, care should be taken that no data that compromises the integrity of the individual is being stored (let alone being made accessible to the public). However, our examples also show that even the mining of data at the aggregated level of user masses is able to reveal valuable information from which the individual can benefit. We thus believe that the collection of usage data should not per se be demonized. Rather, we should carefully weight pros and cons before storing any piece of information in order to find the best trade-off between privacy considerations and the benefits of the end users.

Bibliography

[1] R. Agrawal, A. V. Evfimievski, and R. Srikant. Information Sharing Across Private Databases. In *SIGMOD Conference*, pages 86–97, 2003.

[2] M. B. Alonso and D. V. Keyson. Musiccube: making digital music tangible. In *CHI Extended Abstracts*, pages 1176–1179, 2005.

[3] R. Atterer, M. Wnuk, and A. Schmidt. Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 203–212, 2006.

[4] J. Aucouturier and F. Pachet. Music Similarity Measures: What's the Use? In *ISMIR*, 2002.

[5] J. Aucouturier and F. Pachet. Scaling up Music Playlist Generation. In *ICME*, 2002.

[6] J. Aucouturier and F. Pachet. Representing musical genre: A state of the art. *Journal of New Music Research*, 32(1):83–93, 2003.

[7] J. Aucouturier and F. Pachet. Improving timbre similarity: How high is the sky. *Journal of Negative Results in Speech and Audio Sciences*, 1(1):1–13, 2004.

[8] C. Baccigalupo and E. Plaza. A case-based song scheduler for group customised radio. In *ICCBR*, 2007.

[9] D. Bainbridge, S. Cunningham, and J. Downie. How people describe their music information needs: A grounded theory analysis of music queries. In *ISMIR*, pages 221–222, 2003.

[10] A.-L. Barabási, H. Jeong, Z. Néda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A*, 311(4):590–614, 2002.

[11] F. Bentley and C. J. Metcalf. Sharing motion information with close family and friends. In *CHI*, 2007.

[12] F. Bentley, C. J. Metcalf, and G. Harboe. Personal vs. commercial content: the similarities between consumer use of photos and music. In *CHI*, pages 667–676, 2006.

[13] A. Berenzweig, B. Logan, D. Ellis, and B. Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

[14] J. Bergman, J. Kauko, and J. Keränen. Hands on music: physical approach to interaction with digital music. In *Mobile HCI*, 2009.

[15] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *ESA*, 2003.

[16] T. Brants. Test data likelihood for plsa models. *Inf. Retr.*, 8(2):181–196, 2005.

[17] M. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney. Content-based music information retrieval: current directions and future challenges. *PROCEEDINGS-IEEE*, 96(4):668, 2008.

[18] S. Counts. Group-based mobile messaging in support of the social side of leisure. *Computer Supported Cooperative Work*, 16(1-2), 2007.

[19] G. Cselle, K. Albrecht, and R. Wattenhofer. Buzztrack: topic detection and tracking in email. In *IUI*, 2007.

[20] M. R. David Gleich, Leonid Zhukov and K. Lang. The World of Music: SDP layout of high dimensional data. In *InfoVis*, 2005.

[21] V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *NIPS*, pages 705–712, 2002.

[22] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *JASIST*, 41(6), 1990.

[23] P. Dodds, R. Muhamad, and D. Watts. An experimental study of search in global social networks. *Science*, 301(5634):827–829, August 2003.

[24] S. Dornbush, J. English, T. Oates, Z. Segall, and A. Joshi. XPod: A human activity aware learning mobile music player. In *Proceedings of the Workshop on Ambient Intelligence, IJCAI*, volume 331, 2007.

[25] J. Downie. Establishing Music Information Retrieval (MIR) and Music Digital Library (MDL) Evaluation Frameworks: Preliminary Foundations and Infrastructures. *The MIR/MDL Evaluation Project White Paper Collection Edition# 3*, page 3, 2003.

[26] J. S. Downie and S. J. Cunningham. Toward a theory of music information retrieval queries: System design implications. In *ISMIR*, 2002.

[27] N. Eagle and A. Pentland. Social Serendipity: Mobilizing Social Software. *IEEE Pervasive Computing*, 4(2):28–34, 2005.

[28] D. P. W. Ellis, B. Whitman, A. Berenzweig, and S. Lawrence. The quest for ground truth in musical artist similarity. In *ISMIR*, 2002.

[29] E. Elmacioglu and D. Lee. On Six Degrees of Separation in DBLP-DB and More. *SIGMOD Rec.*, 34(2):33–40, 2005.

[30] S. Farnham and P. Keyani. Swarm: Hyper awareness, micro coordination, and smart convergence through mobile group text messaging. In *HICSS*, 2006.

[31] S. Farnham, W. Portnoy, A. Turski, L. Cheng, and D. Vronay. Personal map: Automatically modeling the user's online social network. In *INTERACT*, 2003.

[32] M. Fernández, D. Vallet, and P. Castells. Probabilistic Score Normalization for Rank Aggregation. In *ECIR*, pages 553–556, 2006.

[33] S. Fickas, G. Kortuem, J. Schneider, Z. Segall, and J. Suruda. When cyborgs meet: Building communities of cooperating wearable agents. In *Proc. Intl. Symp. Wearable Computers, IEEE CS Press, Los Alamitos, Calif*, pages 124–132, 1999.

[34] J. Foote. Content-based retrieval of music and audio. In *Proceedings of SPIE*, volume 3229, page 138, 1997.

[35] J. Frank, T. Lidy, P. Hlavac, and A. Rauber. Map-based music interfaces for mobile devices. In *ACM Multimedia*, 2008.

[36] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, pages 1–19, 2004.

[37] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. *Comput. Geom.*, 29(1):3–18, 2004.

[38] E. Garfield. Citation Analysis as a Tool in Journal Evaluation. *Science*, (178):471–479, 1972.

[39] M. Girvan and M. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12), 2002.

[40] S. Gregory. An algorithm to find overlapping community structure in networks. In *PKDD*, 2007.

[41] S. Gregory. A fast algorithm to find overlapping communities in networks. In *ECML/PKDD*, 2008.

[42] I. Guy, I. Ronen, and E. Wilcox. Do you know?: recommending people to invite into your social network. In *IUI*, 2009.

[43] D. Harel and Y. Koren. Graph Drawing by High-Dimensional Embedding. *Graph Drawing*, pages 207–219, 2002.

[44] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[45] D. B. Hauver and J. C. French. Flycasting: On the fly broadcasting. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.

[46] O. Hilliges, P. Holzer, R. Klüber, and A. Butz. Audioradar: A metaphorical visualization for the navigation of large music collections. In *Smart Graphics*, 2006.

[47] A. Hinneburg, D. A. Keim, and M. Wawryniuk. Hd-eye - visual clustering of high dimensional data. In *ICDE*, 2003.

[48] T. Hirsch and J. Henry. Txtmob: text messaging for protest swarms. In *CHI Extended Abstracts*, 2005.

[49] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[50] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42(1):177–196, 2001.

[51] S. Hohenberger and S. A. Weis. Honest-Verifier Private Disjointness Testing Without Random Oracles. In *Privacy Enhancing Technologies*, pages 277–294, 2006.

[52] B. A. Huberman, M. K. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *ACM Conference on Electronic Commerce*, pages 78–86, 1999.

[53] Y. Iwatani. Love: Japanese Style. *Wired News*, 11, 1998.

[54] A. Kiayias and A. Mitrofanova. Testing Disjointness of Private Datasets. In *Financial Cryptography*, pages 109–124, 2005.

[55] R. Kikin-Gil. Affective is effective: how information appliances can mediate relationships within communities and increase one's social effectiveness. *Personal and Ubiquitous Computing*, 10(2-3), 2006.

[56] J. Kleinberg. The small-world phenomenon: an algorithmic perspective. In *STOC*, pages 163–170, 2000.

[57] J. Kleinberg. Small-world phenomena and the dynamics of information. In *Advances in neural information processing systems: proceedings of the 2002 conference*, page 431. MIT Press, 2002.

[58] P. Knees, M. Schedl, T. Pohle, and G. Widmer. An Innovative Three-Dimensional User Interface for Exploring Music Collections Enriched with Meta-Information from the Web. In *ACM Multimedia*, pages 17–24, 2006.

[59] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):87, 1997.

[60] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. Structure and evolution of blogspace. *Commun. ACM*, 47(12):35–39, 2004.

[61] P. Lamere. Social tagging and music information retrieval. *Journal of New Music Research*, 37(2):101–114, 2008.

[62] E. Law, L. Von Ahn, R. Dannenberg, and M. Crawford. Tagatune: A game for music and sound annotation. In *International Conference on Music Information Retrieval (ISMIR07)*, pages 361–364. Citeseer, 2003.

[63] B. Lee, M. Czerwinski, G. Robertson, and B. B. Bederson. Understanding Research Trends in Conferences using PaperLens. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1969–1972, New York, NY, USA, 2005. ACM Press.

[64] J. H. Lee and J. S. Downie. Survey of music information needs, uses, and seeking behaviours: Preliminary findings. In *ISMIR*, 2004.

[65] S. Leitich and M. Topf. Globe of music - music library visualization using geosom. In *ISMIR*, 2007.

[66] M. Levy and M. Sandler. A Semantic Space for Music Derived from Social Tags. In *International Conference on Music Information Retrieval (ISMIR07)*, 2007.

[67] K. Lewis, J. Kaufman, M. Gonzalez, A. Wimmer, and N. Christakis. Tastes, ties, and time: A new social network dataset using Facebook.com. *Social Networks*, 30(4), 2008.

[68] Y. Li, J. Tygar, and J. Hellerstein. Private matching. *Computer Security in the 21st Century*, pages 25–50, 2005.

[69] D. Liben-Nowell, J. Novak, R. Kumar, P. Raghavan, and A. Tomkins. Geographic routing in social networks. *Proceedings of the National Academy of Sciences*, 102(33):11623–1162, 2005.

[70] A. Linde. On the pitfalls of journal ranking by impact factor. *Eur J Oral Sci*, 106(1):525–526, February 1998.

[71] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[72] X. Liu, J. Bollen, M. L. Nelson, and H. V. de Sompel. Co-authorship networks in the digital library research community. *Inf. Process. Manage.*, 41(6):1462–1480, 2005.

[73] B. Logan. Content-based playlist generation: Exploratory experiments. In *ISMIR*, 2002.

[74] B. Logan and A. Salomon. A music similarity function based on signal analysis. In *ICME*, 2001.

[75] D. Lübbers. Sonixplorer: Combining visualization and auralization for content-based exploration of music collections. In *ISMIR*, 2005.

[76] M. H. MacRoberts and B. R. MacRoberts. Problems of citation analysis: A critical review. *JASIS*, 40(5):342–349, 1989.

[77] M. Mandel and D. Ellis. A web-based game for collecting music metadata. *Journal of New Music Research*, 37(2):151–165, 2008.

[78] Y. Matsuo, J. Mori, M. Hamasaki, K. Ishida, T. Nishimura, H. Takeda, K. Hasida, and M. Ishizuka. Polyphonet: an advanced social network extraction system from the web. In *WWW*, pages 397–406, 2006.

[79] M. McLuhan. *Understanding Media: the Extensions of Man.* pub-MCGRAW-HILL, 1964. Reprinted by MIT Press, Cambridge, MA, 1994.

[80] F. Menczer. Growing and navigating the small world web by local content. *Proceedings of the National Academy of Sciences*, 99(22):14014, 2002.

[81] F. Mörchen, A. Ultsch, M. Nöcker, and C. Stamm. Databionic visualization of music collections according to perceptual distance. In *ISMIR*, 2005.

[82] M. A. Nascimento, J. Sander, and J. Pound. Analysis of SIGMOD's Co-Authorship Graph. *SIGMOD Rec.*, 32(3):8–10, 2003.

[83] R. Neumayer, M. Dittenbach, and A. Rauber. PlaySOM and PocketSOMPlayer, Alternative Interfaces to Large Music Collections. In *ISMIR*, pages 618–623, 2005.

[84] M. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3), 2006.

[85] M. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23), 2006.

[86] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2), 2004.

[87] M. E. J. Newman. Who is the best connected scientist? A study of scientific coauthorship networks. *Physical Review E*, 64(1):016132–1–016132–7, July 2001. Scientific collaboration networks. Part II. Shortest paths, weighted networks, and centrality.

[88] V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending modularity definition for directed graphs with overlapping communities. 2008.

[89] Nokia. plazes.com. www.plazes.com, 2009. [Online; accessed 13-January-2009].

[90] A. Oulasvirta, M. Raento, and S. Tiitta. ContextContacts: redesigning SmartPhone's contact book to support mobile awareness and collaboration. In *Mobile HCI*, 2005.

[91] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1998.

[92] E. Pampalk, S. Dixon, and G. Widmer. On the evaluation of perceptual similarity measures for music. In *DAFx*, 2003.

[93] E. Pampalk, A. Flexer, and G. Widmer. Improvements of audio-based music similarity and genre classification. In *ISMIR*, volume 5. Citeseer, 2005.

[94] E. Pampalk and M. Goto. Musicrainbow: A new user interface to discover artists using audio-based similarity and web-based labeling. In *ISMIR*, 2006.

[95] E. Pampalk, T. Pohle, and G. Widmer. Dynamic playlist generation based on skipping behavior. In *ISMIR*, pages 634–637, 2005.

[96] P. Persson and Y. Jung. Nokia sensor: from research to product. In *Designing for User eXperience*. AIGA: American Institute of Graphic Arts New York, NY, USA, 2005.

[97] J. Platt, C. Burges, S. Swenson, C. Weare, and A. Zheng. Learning a Gaussian Process Prior for Automatically Generating Music Playlists. *NIPS*, 14:1425–1432, 2002.

[98] T. Pohle, E. Pampalk, and G. Widmer. Generating similarity-based playlists using traveling salesman algorithms. In *DAFx*, 2005.

[99] R. Ragno, C. J. C. Burges, and C. Herley. Inferring similarity between music objects with application to playlist generation. In *MIR*, pages 73–80, 2005.

[100] S. Robertson. Understanding Inverse Document Frequency: On theoretical arguments for IDF. *Journal of Documentation*, (5):503–520, 2004.

[101] G. Salton, E. A. Fox, and H. Wu. Extended Boolean information retrieval. *Commun. ACM*, 26(11):1022–1036, 1983.

[102] O. Sandberg. Distributed routing in small-world networks. In *ALENEX*, 2006.

[103] M. Sarkar and M. H. Brown. Graphical fisheye views. *Commun. ACM*, 37(12):73–83, 1994.

[104] A. Scharnhorst and M. Thelwall. Citation and hyperlink networks. *Current Science*, 89(9), November 2005.

[105] A. Schmidt. Implicit Human Computer Interaction Through Context. *Personal and Ubiquitous Computing*, 4(2/3), 2000.

[106] J. Seo and B. Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization*, 4(2), 2005.

[107] B. Shneiderman. Extreme visualization: squeezing a billion records into a million pixels. In *SIGMOD Conference*, 2008.

[108] M. Slaney and W. White. Similarity based on rating data. In *Proc. of Int. Symposium on Music Information Retrieval.* Citeseer, 2007.

[109] A. Smeaton, T. Sodring, K. McDonald, G. Keogh, and C. Gurrin. Analysis of Papers from Twenty-Five Years of SIGIR Conferences: What Have We Been Doing for the Last Quarter of a Century? *SIGIR Forum*, 36(2,), 2002.

[110] I. E. Smith, S. Consolvo, A. LaMarca, J. Hightower, J. Scott, T. Sohn, J. Hughes, G. Iachello, and G. D. Abowd. Social disclosure of place: From location technology to communication practices. In *Pervasive*, 2005.

[111] R. Snodgrass. Journal relevance. *SIGMOD Rec.*, 32(3):11–15, 2003.

[112] J. C. Tang, N. Yankelovich, J. Begole, M. V. Kleek, F. C. Li, and J. R. Bhalodia. Connexus to awarenex: extending awareness to mobile users. In *CHI*, 2001.

[113] M. Terry, E. Mynatt, K. Ryall, and D. Leigh. Social net: using patterns of physical proximity over time to infer shared interests. In *CHI'02 extended abstracts on Human factors in computing systems*, pages 816–817, 2002.

[114] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

[115] M. Torrens, P. Hertzog, and J. L. Arcos. Visualizing and exploring personal music libraries. In *ISMIR*, 2004.

[116] E. Tsunoo, G. Tzanetakis, N. Ono, and S. Sagayama. Audio genre classification using percussive pattern clustering combined with timbral features. In *Proc. of ICME*, pages 382–385, 2009.

[117] D. Turnbull, L. Barrington, and G. R. G. Lanckriet. Five approaches to collecting tags for music. In *ISMIR*, pages 225–230, 2008.

[118] G. Tzanetakis. Marsyas3D: A Prototype Audio Browser-Editor using a Large Scale Immersive Visual and Audio Display. In *ICAD*, 2001.

[119] G. Tzanetakis and P. R. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002.

[120] R. van Gulik and F. Vignoli. Visual playlist generation on the artist map. In *ISMIR*, pages 520–523, 2005.

[121] F. van Ham, J. van Wijk, and T. Eindhoven. Interactive Visualization of Small World Graphs. *InfoVis*, pages 199–206, 2004.

[122] F. Vignoli, R. van Gulik, and H. van de Wetering. Mapping music in the palm of your hand, explore and discover your collection. In *ISMIR*, 2004.

[123] S. Wasserman, K. Faust, and D. Iacobucci. *Social Network Analysis : Methods and Applications.* Cambridge University Press, 1994.

[124] D. J. Watts, P. S. Dodds, and M. Newman. Identity and Search in Social Networks. *Science*, 296:1302–1305, 2002.

[125] D. J. Watts and S. H. Strogatz. Collective Dynamics of "Small-World" Networks. *Nature*, 393:440–442, 1998.

[126] B. Whitman and S. Lawrence. Inferring descriptions and similarity for music from community metadata. In *Proceedings of the 2002 International Computer Music Conference*, pages 591–598. Citeseer, 2002.

[127] J. Yang, M. O. Ward, E. A. Rundensteiner, and S. Huang. Visual hierarchical dimension reduction for exploration of high dimensional datasets. In *VisSym*, 2003.

[128] A. Yao. Protocols for secure computations. *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[129] H. Zhang, A. Goel, and R. Govindan. Using the small-world model to improve freenet performance. *Comput. Netw.*, 46(4):555–574, 2004.

[130] N. Zhang and W. Zhao. Distributed Privacy Preserving Information Sharing. In *VLDB*, pages 889–900, 2005.

Curriculum Vitae

February 12, 1979	Born in Basel, Switzerland
1986–1999	Primary, secondary, and high schools in Flüh-Hofstetten/SO and Oberwil/BL, Switzerland
1999–2004	Studies in information technologies and electrical engineering, ETH Zurich, Switzerland
October 2004	M.Sc. in information technologies and electrical engineering, ETH Zurich, Switzerland
2004–2005	Risk controlling, Winterthur Insurances, Switzerland
2005–2010	Ph.D. student, research and teaching assistant, Distributed Computing Group, Prof. Roger Wattenhofer, ETH Zurich, Switzerland
August 2010	Ph.D. degree, Distributed Computing Group, ETH Zurich, Switzerland Advisor: Prof. Roger Wattenhofer Co-examiner: Prof. Albrecht Schmidt, University Duisburg-Essen, Germany

Publications

The following list contains all publications I have co-authored during my time as a Ph.D. Student at the Distributed Computing Group at ETH Zurich:

1. Social Audio Features for Advanced Music Retrieval Interfaces. Michael Kuhn, Roger Wattenhofer, and Samuel Welten. *ACM Multimedia*, October 2010.

2. Improving Personal Diaries Using Social Audio Features. Michael Kuhn, Roger Wattenhofer, and Samuel Welten. *Google Grand Challenge @ ACM Multimedia*, October 2010.

3. Visually and Acoustically Exploring the High-Dimensional Space of Music. Lukas Bossard, Michael Kuhn, and Roger Wattenhofer. *IEEE International Conference on Social Computing (SocialCom)*, August 2009.

4. Cluestr: Mobile Social Networking for Enhanced Group Communication. Reto Grob, Michael Kuhn, Roger Wattenhofer, and Martin Wirz. *International Conference on Supporting Group Work (GROUP)*, May 2009.

5. From Web to Map: Exploring the World of Music. Olga Goussevskaia, Michael Kuhn, Michael Lorenzi, and Roger Wattenhofer. *IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, December 2008.

6. VENETA: Serverless Friend-of-Friend Detection in Mobile Social Networking. Marco von Arb, Matthias Bader, Michael Kuhn, and Roger Wattenhofer. *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, October 2008.

7. Exploring Music Collections on Mobile Devices. Olga Goussevskaia, Michael Kuhn, and Roger Wattenhofer. *International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI)*, September 2008.

8. Distributed Asymmetric Verification in Computational Grids. Michael Kuhn, Stefan Schmid, and Roger Wattenhofer. *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2008.

9. The Layered World of Scientific Conferences. Michael Kuhn and Roger Wattenhofer. *Asia Pacific Web Conference (APWeb)*, April 2008.

10. The Theoretic Center of Computer Science Michael Kuhn and Roger Wattenhofer. (Invited paper) *SIGACT News Volume 38, Number 4, (Whole Number 145)*, December 2007.

11. Layers and Hierarchies in Real Virtual Networks. Olga Goussevskaia, Michael Kuhn, and Roger Wattenhofer. *IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, November 2007.

12. Community-Aware Mobile Networking. Michael Kuhn and Roger Wattenhofer. *Workshop on Mobile Services and Personalized Environments (MSPE)*, November 2006.