Recommendation And Visualization Techniques For large Scale Data

Afshin Moin

ANNÉE 2012

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale Matisse

présentée par

Afshin Moin

préparée à l'unité de recherche IRISA – UMR 6074
Institut de Recherche en Informatique et Systèmes Aléatoires
ISTIC

Recommendation And Visualization Techniques For Large Scale Data

Thèse soutenue à Rennes
le 9 Juillet 2012

devant le jury composé de :

Pierre FRAIGNIAUD
DR CNRS / rapporteur

Marie-Christine ROUSSET
Prof. Univ. Grenoble 1 / rapporteur

Eric FLEURY
Prof. ENS de Lyon / examinateur

Arnaud GUYADER
MdC Univ. Rennes 2 / examinateur

Anne-Marie KERMARREC
DR INRIA, Rennes / directrice de thèse

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all of those many people without whom this work would not have been possible. I wish to show my greatest appreciation to my supervisor Prof. Anne-Marie Kermarrec whose support and encouragement was instrumental in the accomplishment of this degree. Deepest gratitude are also due to Prof. Pierre Fraigniaud and Prof. Marie-Christine Rousset who accepted to report on this document, and to jury members Prof. Eric Fleury and Dr. Arnaud Guyader. I also thank my dear colleague Dr. Davide Frey for his valuable comments and fruitful discussions. Last but not least, I am deeply grateful to my beloved family for their endless love and constant support through the whole duration of my studies.

Afshin Moin
9 July 2012

CONTENTS

LIST OF FIGURES

List of Tables

CHAPTER 1

INTRODUCTION

Thanks to the Internet, data, services and applications are now accessible from everywhere. If twenty years ago, it was not even imaginable to find the recipes of Chinese cuisine or the local information about an exotic vacation destination, it is nowadays as fast as a simple search query to find tones of information about any favorite subject in the finest details. As a natural consequence of these progresses, the volume of the data generated in different applications has grown with an unprecedented rate. Furthermore, new perspectives have been opened on how the Web technology can influence our lifestyle: with the advent of Web 2.0, users can now contribute to the content of the Web. This has given rise to new applications like social networking and collaborative content sharing . With more than 800 millions of users as of July 2011, Facebook has not still stopped from growing. Internet users share everything, ranging from everyday life events to services and applications, on their Facebook account. Similarly, they rely on websites like YouTube and Flickr to diffuse their content, express their opinions and get informed of the opinions of the others.

Following these remarkable milestones, we are now confronted with new challenges to efficiently process and represent the huge mass of data at our disposal. While availability of the information is no more a problem, the volume of data has become so immense that it is getting overwhelmingly hard to find relevant content with satisfactory quality. Consequently, users are in need of help in the process of decision-making for finding content matching properly their needs. The task of mining the data to offer personalized content to users is known as *recommendation*. However, finding relevant content is not the whole problem, as the results also need to be presented in an informative way. Efficient representation of the data is of great importance both to increase the utility of the information for the users, and to develop efficient tools for data analysis.

This document is centered around the two axes of recommending relevant content and its proper visualization. In our study, we will concentrate on *collaborative filtering* as the most successful approach to recommender systems. Meanwhile, we focus on the problem of *graph drawing*, as most types of data can be modeled as graphs. We also show how the same techniques used in recommendations can be modified to consider visualization requirements. We first provide an introduction to recommender systems in Section 1.1 and graph drawing in Section 1.2 to clarify the goals and

objectives in these domains. Technical background on each topic is presented in the moment in the corresponding chapter. The exact contributions and the road map are outlined in Section 1.3.

1.1 Recommender Systems

The problem of mining relevant information from among the large mass of data has given rise to a popular research domain known as *Recommender Systems* (RS). More precisely, recommender systems address all the information filtering techniques attempting to find items matching the needs of an active user, in order to provide her with *personalized* recommendations. Items can be webpages, multimedia material like movies and music, news, scientific articles, books, travel packages, restaurants or any other object on the Web. The notion of personalization has a central role in the definition of recommender systems. Of course, in a very simplistic scenario, it is possible to compute a general ranking for items, and recommend all users with some popular ones. Nevertheless, almost all the literature of RS is exclusively devoted to personalized recommendations. The domain of RS is closely related to machine learning, data mining and information retrieval. As a consequence, many techniques in the design of RS have been adopted from these domains. Recommender systems have recently enjoyed great popularity both in industry and in academics. All major players of the online commerce market (e.x. Amazon, eBay, YouTube, Netflix, MovieLens, Last.fm, Pandora, etc.) have integrated recommender engines in their websites, and are actively working to enhance their quality. Indeed, recommender systems have been proved to be an undeniable asset for increasing user satisfaction and to grow the clientale. The attention to RS grew significantly following the Netflix Prize competition [61, 9, 99]. The $1M\$$ prize was about improving the precision of the existing recommender algorithm of Netflix called Cinematch by 10%. It was finally won by a coalition of three research groups in September 2009. The winners published a summary of their approach in [77, 124, 106].

The primary goal of an RS is to estimate the interest of users for items. Though, RS can do more than prediction both for the service providers and for the users [17]. Recommender systems can also win the trust of the users, in case they provide them with convincing recommendations. Trust is an important factor in making the clients loyal to the website. The data gathered by an RS is a precious source of information for market analysis and learning the client preferences. In addition, users may use an RS to become more aware of their preferences and/or the existing items. In this case, the RS is exploited the same way a search engine is used. Users can also take advantage of recommender systems to express their opinions, or to try to influence the other users.

The design, implementation and evaluation of RS depend on the requirements of the underlying application, type of data at hand and the objectives of the RS. Some applications may have special requirements. For instance, in some contexts one item may not be consumed more than once: a movie RS must no longer recommend a movie the user has already watched. On the contrary, a music RS can keep on recommending the same item multiple times, as music fans usually listen to their favorite songs more than once.

1.1.1 Input data

Recommender systems rely on the input data from users' previous activities in order to recommend relevant content to users. The input data can be either in implicit or explicit form. The explicit feedback is the more reliable type of data which is issue of the direct report of users about their

experience. Yet, explicit feedback is generally hard to collect, as many users prefer not to disclose their opinions for time and privacy matters. Implicit data is another type of the input which, unlike the explicit feedback, is easy to collect. Any user activity like visiting a website, clicking on a link, skipping a song, etc. is categorized as implicit feedback. However, the implicit feedback is noisy compared to the explicit feedback, as it does not necessarily report the correct opinion of a user about an item. For example, a user may have clicked on the advertisement of a movie, but not have liked it after its watching. In this case, the RS misinterprets this activity as positive feedback, while this interpretation is false in reality. In order to gather the more reliable explicit feedback, most websites ask their users to rate the items or to fill in client satisfaction questionnaires. Feedback may be gathered in various scales. Implicit feedback and like/dislike buttons are examples of a binary scale. Nevertheless, the most common is the five star rating scale, where users grant items from one to five stars according to their level of satisfaction from the corresponding item. Another way of taking user feedback is to let the users tag the items. This strategy has been widely applied in collaborative tagging websites like delicious. Tags and ratings give very different insight into the properties on an item. While tags bear information about the *content* of items, ratings deal with their *quality*.

1.1.2 Requirements of Recommender Algorithms

The utility of an RS for users and for the service providers depends on multiple criteria. No RS can meet optimally all of these criteria, as they are usually conflicting. Consequently, a tradeoff must be made between them. Here, we list the most important requirements of an RS.

Accuracy is undoubtedly one of the most important criteria an RS must meet. It addresses how precisely an RS can estimate the interest of users for items. Higher accuracy means higher user satisfaction, better commerce rate and more successful prediction of the incoming trends. A legacy of the Netflix Prize competition, was to concentrate most research on designing more accurate recommender systems. Although accuracy is an important quality measure of recommendations, the utility of an RS is determined by an ensemble of factors going beyond its accuracy.

Scalability In recommendation applications, the number of users and items is continuously growing. As a consequence, it is important that recommender algorithms scale well with the number of users and items. In the majority of applications, the number of users is larger and grows faster than the number of items. For instance, the number of users in the Netflix dataset is about half a million while the number of items (i.e. movies in this example) is less than 18000. This implies that scalability with respect to users is more advantageous than scalability with respect to items.

Explainability It is important that a recommender system can explain the reason behind the recommendations. Explainability has an important effect in convincing the users to accept the recommendations. It is in general easy to explain the recommendations based on similar items, because users know for sure the items they have consumed before. For instance, if an Amazon user has added an item, say A, to her shopping card, a similar item, say B, which has received high ratings from other clients, is recommended to her with an explanation like "Item B is recommended because you added item A to your shopping card." An example of Amazon recommender interface is presented in Figure 1.1. On the other hand, explaining recommendations based on similar users is

more challenging, because an RS may exploit the feedback of similar users who do not necessarily know each other to compute the recommendations. In a new line of research known as *social* or *community-based* recommendations [42, 69, 45, 91, 119] the profiles of friends in a user's social network is used to generate recommendations. In this special case, naming these users can be useful to explain the reason behind a recommendation. An example of this type of explanations is phrases like "Alice and Mike have played Poker online, do you want to play too?".

Diversity Users have in general multiple interests. They can also become attracted in new areas or change their interests over the time. It is not reasonable to recommend a user with a list of items, all from the same type, even though the predicted ratings for them are high. Hence, a recommender system must be capable of packing items of diverse types in the recommendation list of a user, delegating her different interests.

Novelty is about the originality of the recommendations. It is nice for example that fans of a singer find the new album of their favorite artist as soon as it comes out. Although novelty is an asset for an RS, it is somehow risky to recommend novel items, as not enough information is yet available to estimate their quality with high confidence. Hence, emphasizing on novelty of recommendations can lead to a drop in the accuracy.

Serendipity It is nice that users can find interesting items by chance with the help of the RS. This property is called serendipity, and is closely related to diversity, but is about those items that the user would not think of their consuming on her own. In the example of the new album of a signer, the recommendation is novel but not serendipitous as the user could follow her favorite signer.

Presentation The simplest and at the same time the most common way of representing the recommendations is to offer them as a ranked list. Nevertheless, the correlation between items in a ranked list may remain vague to users. The role of an RS is to help users in the process of decision-making, when they have limited knowledge of the underlying domain. Consequently, delivering the recommendations as a list is an overly simplistic way for representation of the results. In fact, this immature type of representation can waste a good part of knowledge that have already been mined by the RS, as will be discussed in detail in Section 4.

1.1.3 Recommendation Techniques

Different approaches have been invented to build recommender systems. The most important of them are presented in the sequel.

Knowledge-based recommenders suggest items matching the needs of a user [14, 120, 16, 37, 136], once she has determined the features of her favorite item. This type of recommendation is useful when exact information about the items (like price, type, color, etc.) is available as a feature vector. The system matches the favorite attributes of the user with item attributes to find and recommend those items meeting user preferences the best.

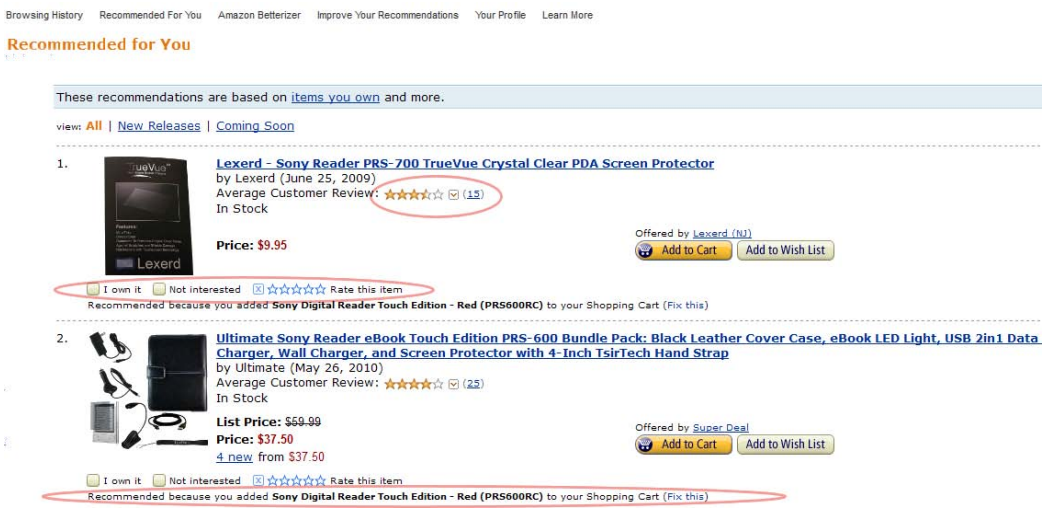


Figure 1.1: An example of Amazon recommendations.

Demographic recommenders categorize users into demographic groups based on their social and personal attributes like age, gender, address, etc. Users are then recommended with items having been interesting to users of the same category. Some instances of such systems are presented in [66, 82]. A drawback associated with demographic recommenders is the difficulty of gathering the demographic data of users, as many of them may be unwilling to disclose their private information or even lie about that. Furthermore, gathering such information may be criticized of jeopardizing users' privacy.

Content-based recommenders exploit the content information of items to estimate their similarity. In this approach, content information must be available as Meta data. Items similar to those in the active user's profile are then recommended to her. Examples of these systems are [62, 4, 105, 27, 97]. In some contexts like musical recommendations, content-based recommenders have been very successful. Pandora is an example of this success. It is a music recommendation service, where a user first needs to provide the system with a number of songs or artists they like. The RS then offers further songs from the favorite genres of the user. Users will have the possibility of giving their feedback through rating the songs. The ratings are taken into account for computing the future recommendations. Content-based recommendations enjoy good explainability, as it is easy to tell the users that they are recommended with an item because it is similar to their previous purchases. For example, the Amazon client in Figure 1.1 has searched for a Sony book reader. She is then recommended with a screen protector for that. One important drawback of content-based approach is the difficulty of gathering content information of the items. Namely, this involves the expensive and time-consuming task of adding meta data to the items, sometimes requiring expert taggers. Content-based approach also suffers from lack of novelty and serendipity, as all recommended items are very similar to the items a user has already searched or consumed. Furthermore, the precision of this approach is in general less than the more popular approach of collaborative filtering discussed next.

Collaborative filtering is the prevalent approach to recommender systems. In this approach, each user is recommended with the items having been interesting to the users with similar profiles. The reason behind the popularity of this method resides in some of its advantages like high precision, and no need to content information of the items. Users give more value to the recommendations based on other users' experience than taking a list of items with relevant content [93, 92] - as is done in content-based recommenders - because similarity in content does not necessarily guarantee the good quality of the recommended items. In this document, we concentrate on collaborative filtering techniques. They are classified in two general categories of memory-based and model-based schemes. The memory-based approach (also known as the neighborhood-based approach) uses directly the history of user activities to estimate the similarity between users or items. A missing rating is estimated by averaging either the ratings the item has received from users with similar history to the active user (user-based approach), or the ratings the active user has given to items similar to the corresponding item (item-based approach). Model-based approaches use the rating history to learn a model for prediction. Model parameters are learnt automatically from the data through global optimization methods. Many models have been suggested for collaborative filtering. Some examples include matrix factorization based on Singular Value Decomposition [76, 78, 8, 103], graph-based methods [59, 95], information theoretic approaches based on maximum entropy [65, 137, 104], boltzmann machines [113], latent dirichlet allocation [12] and latent semantic analysis [56].

Hybrid recommenders It is common to use different recommendation techniques together to alleviate the drawbacks of one with the advantages of the others [17, 107, 6]. The most common is to merge content-based and collaborative filtering strategies [94, 22, 107]. When a user is new to the system, and has consumed very few items, her similarity with other users cannot be estimated with confidence. Hence, generating content-based recommendations can be considered as a solution to alleviate this lack of information in the phase of cold start. Likewise, hybrid recommenders can be built by merging collaborative filtering and social recommendation schemes, or even mixing different techniques of the same type.

1.1.4 Evaluation of Recommender Systems

Evaluation of RS has always been challenging. One reason for this difficulty is that it is hard to know the users' opinion about their recommendations. Some websites like Amazon ask their users' feedback in this regard. In addition, while some aspects of recommender systems like accuracy can be measured relatively easily, some more subjective goals like serendipity are more challenging to measure. Since our concentration will be on collaborative filtering, we discuss the evaluation methods and metrics of this technique. A detailed overview of the goals and evaluation methods of collaborative filtering algorithms is provided by Herlocker et al. [55].

RS can be evaluated online or offline. The online evaluation consists of monitoring the reaction of users to their recommendations. This can only be done by service providers, preventing researchers from comparing their algorithms. Furthermore, the results will depend on the test users exploiting the system during the evaluation time. As a consequence, the experiments are not repeatable. Most often, recommender systems are evaluated offline. The common method for offline evaluation is cross validation. In this method a dataset of user activities is picked up. In a collaborative filtering recommender, where feedback is in the form of ratings, this dataset would consist of a set of users and items, together with the ratings they have given to the items. The dataset is then divided into two

disjoint subsets: training set and test set. The algorithm is optimized on the training set. Predictions are made for the test set, and compared against the real values of the ratings. This approach allows for repeatable and comparable experiments. One popular measure to estimate the total prediction error of an RS is the Mean Absolute Error (MAE):

$$\text{MAE} = \frac{\sum_{r_{ui} \in R_{Test}} |\hat{r}_{ui} - r_{ui}|}{|R_{Test}|}, \tag{1.1}$$

where r_{ui} is the rating of user u for item i. R_{Test} is the set of all ratings in the test set, and $\hat{r}_{ui}$ the estimation of the algorithm for r_{ui}. MAE treats equally all values of error. Considering the fact that consumption of each item incurs cost to the users, it is very disappointing to receive recommendations with large error. For example, if a user has bought a book which has been recommended to her with 5 stars, but finds out after its reading that it would merit only 2 stars, she would probably take no more advices from the RS. Consequently, it is better to penalize larger errors more than smaller ones. This objective can be achieved by applying the Root Mean Squared Error (RMSE).

$$\text{RMSE} = \sqrt{\frac{\sum_{r_{ui} \in R_{Test}} (\hat{r}_{ui} - r_{ui})^2}{|R_{Test}|}}. \tag{1.2}$$

Following the Netflix Prize, RMSE become the most popular measure for reporting the error of a recommender algorithm. To conform with this trend, we also measure the error of our algorithms using RMSE.

Another important measure of usefulness for recommender systems is *coverage*. It is defined as the proportion of missing ratings in the database for which the recommender algorithm is capable of predicting a value:

$$\text{coverage} = \frac{\left|\hat{R}_{Test}\right|}{|R_{Test}|}, \tag{1.3}$$

where $\hat{R}_{Test}$ is the set of predictable missing ratings.

MAE and RMSE are proper measures to estimate the global prediction error of an RS. If the recommendation function is to recommend a list of relevant items, *recall* and *precision* are the two metrics used most often. In this case, the estimated list of relevant items for each user is compared against the set of items who have been interesting to her in reality. Precision is the fraction of relevant items in users' recommendation lists with respect to the size of recommendations:

$$\text{precision} = \frac{|\text{relevant items} \cap \text{recommended items}|}{|\text{recommended items}|}.$$

Recall is the fraction of relevant items in users' recommendation lists with respect to the number of relevant items:

$$\text{recall} = \frac{|\text{relevant items} \cap \text{recommended items}|}{|\text{relevant items}|}.$$

Having a high recall is not representative of a good RS on its own. It is indeed trivial to have a 100% recall by increasing the size of the recommendation list. A common method is to evaluate both precision and recall together by taking their harmonic mean. This is known as F1 measure:

$$\text{F1} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}.$$

F1 increases sharply if both values increase together. Nevertheless, in some cases it can be misleading in comparison of the algorithms. Specifically, F1 may prefer an algorithm with lower precision and higher recall to a competitor with higher precision and lower recall. Notice that in case an algorithm has low precision and high recall, and the difference between these two is important, it can be interpreted as a spammer rather than a recommender!

To disambiguate the word *precision* in our terminology from the abovementioned definition, whenever we refer to precision in the following chapters, we mean the inverse of RMSE, that is, by higher precision we mean lower RMSE and vice versa.

1.1.5 Challenges against Recommender Systems

Apart from general concerns about algorithmic design, recommender algorithms must deal with special challenges existing in the context of recommender systems. Here, we enumerate some of them.

Sparsity is one the most important problems of recommender systems. The majority of ratings are missing in real-world applications. Considering the large number of items, each user can only rate a small fraction of them. It is therefore of utmost importance that the recommender system processes optimally the data at its disposal.

Cold start An RS must be able to predict ratings for new users and items as soon as possible. This is of course challenging as less is known about the preferences of a user who has just joined the system [1, 117]. Similarly, it is difficult to estimate the ratings of a new item which has been rated by no or very few users. This is known as the problem of cold start, and as already mentioned engages both new users and new items.

Privacy Recommender systems rely on user history to detect similarity patterns. As a consequence, they build datasets which may contain much information about users including their personal information like age and gender and their consuming behavior. Recommender systems are then a good target for hackers. Attacks against RS can appear in different forms like hacking the private information of users, trying to improve or damage the reputation of some items or users, etc. Netflix was accused of breaching users' privacy by releasing the Netflix Prize dataset. Although the dataset was anonymized, two researchers could match its ratings with the ratings of IMDB [98]. The second round of the Netflix Prize was also canceled, following an anonymous user's suing Netflix for privacy breach.

Exploration versus Exploitation From user's point of view, receiving good recommendations is the most important goal of using an RS. This is known as *exploitation* aspect of an RS for the users. One way to have high exploitation is to recommend well-known items to the users. However, recommender systems need to grow their knowledge about *all* items in order to perfection their recommendations and to model better the user profiles. This necessitates acquiring users' feedback about less popular or new items and certain types of products a user has never expressed about. This is called the *exploration* aspect of recommendations. Exploration can have some positive effects on the consuming behavior of the users. For instance, it may initiate a user to a new type of product she has not tried before. Provided the exploration side is not handled in an efficient manner, or

if it is emphasized too much with respect to the exploitation side, the RS becomes embarrassing for the users. As a consequence, an RS needs to make a tradeoff between the exploitation and the exploration aspects. Making such a tradeoff is an important topic of *active learning* [24, 110, 121] algorithms where the goal is to provide a user with precise recommendations by asking her as few questions as possible.

1.2 Graph Drawing

In order for the data to be useful for human users, it must be represented in an understandable way. Proper representation of the data is also central in the design of data analysis tools. Graph drawing is one of the most important branches of information visualization. Almost everything related to computer science can be modeled by graphs. Namely, graph vertices represent the unities of information and edges show the interactions between them. For instance, in social networks users represent the vertices of the graph, and edges denote the existence of some type of interaction (like friendship) between them. The drawing of a graph, called *layout*, is the representation of a graph in 2 or 3 dimensional Euclidean space where vertices and edges are shown with points and curves. Graph layouts are pretty helpful in having graphical insight into the structure of the underlying data.

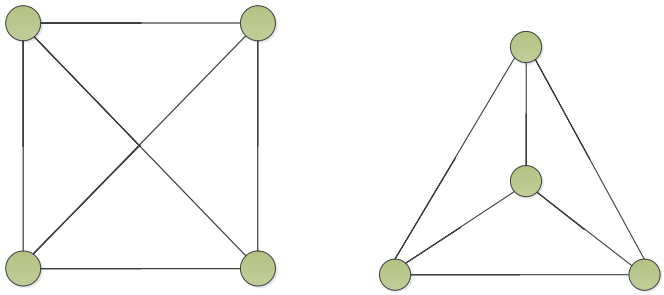
Graph drawing algorithms work based on the intuition that connected edges must lie closer to each other than non-connected vertices. This is equivalent to say that edges must be shorter than non-edges. In the past, when the size of graphs were very small, say about a handful, graphs were drawn by hand. With the unprecedented growth of the size of datasets, this is of course no more feasible. Algorithms are then required to draw large graphs automatically. Graphs are drawn following some conventions. These conventions set general constraints on the geometric representation of edges and vertices. In general, vertices must not overlap in the drawing. There are different conventions about the edges. For undirected graphs, they are as follows:

- Polyline drawing: edges are drawn as polylines between vertices.
- Orthogonal drawing: each edge is consisted of one or more straight vertical or horizontal lines connected to each other with a right angle.
- planar drawing: no edge or vertex crossing is allowed. In this case the drawing is also called the *embedding*. Planar drawing is not possible for most graphs, especially large real-world ones.
- Straight-line drawing: edges are represented with straight lines between connected vertices. In this type of drawing, the problem is reduced to find the coordinates of vertices, and the computation of edges is relaxed.

In this document we are interested in straight-line undirected graph drawing. This is the most popular type of drawing in the literature, and most techniques are about it.

The layout must satisfy some aesthetic criteria in order to be readable. Some widely accepted criteria are:

- Minimum edge crossing: edges must cross each other as few as possible. This enhances the readability of the graph, and allows for easy following of the interactions.



(a) Maximum display of symmetries.

(b) Minimum edge crossing.

Figure 1.2: Conflict between different aesthetic criteria.

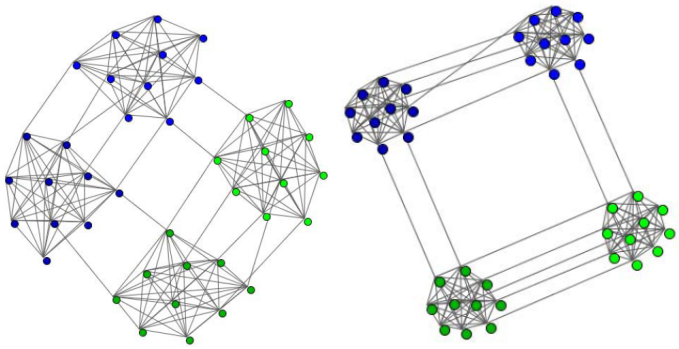
- Uniform edge length: even distribution of edge length increases the clarity of the layout. This property enforces more uniform distribution of connected vertices over the drawing area.
- Uniform vertex distribution: it is important to take maximum advantage of the drawing area. uniform distribution of vertices prevents the case where some areas of the layout are overcrowded while some others remain empty.
- Showing the symmetries: the drawing must reveal as much symmetries existing in the structure of the graph as possible.

It is impossible to achieve optimally more than one criterion, as different criteria seek conflicting objectives. Consequently, models should make a tradeoff between them. For instance, it is shown in Figure 1.2 how displaying the symmetries can conflict the minimum edge crossing.

Force-directed [40, 25, 68, 101, 36, 81] and spectral [74, 19, 80] approaches are the two main approaches to undirected straight-line graph drawing. Force-directed models consider the vertices as a system of interconnected physical particles, and assign attractive and repulsive forces between them. The energy of the system is measured through a scalar function. The optimal layout is found by minim Spectral approaches use the (generalized) eigenvectors of a matrix related to the graph structure like Adjacency or Laplacian as the coordinates. Each approach has its advantages and drawbacks. Force-directed approach is intuitive, relatively easy to implement and generates high quality layouts. Though, it suffers from high running time, difficulty of finding the global minimum of the energy function and lack of strong mathematical analysis. Spectral approaches usually generate layouts with lower quality. However, they have very short running time, and can be analyzed by standard techniques of linear algebra.

1.3 Contribution

In this document we suggest algorithms for collaborative filtering and information visualization. In Chapter 2 we first discuss the decentralized neighborhood-based collaborative filtering algorithms



(a) less abstraction (b) more abstraction

Figure 1.3: Layouts of a graph with 4 clusters in two levels of abstraction.

for P2P systems. Apart from its direct application to contexts like P2P, decentralization is a solution to alleviate the scalability and privacy issues of recommender systems. We apply *epidemic protocols* [126, 31, 11] to decentralize the neighborhood-based recommenders. Epidemic protocols are fully decentralized algorithms coping well with the asynchronous nature of P2P systems. They converge in a few cycles to a neighborhood of similar users. They are robust against churn, and do not require any special recovery protocol in case of failures in the network. Then, a decentralized model-based algorithm based on random walks is proposed as a solution to the problem of sparsity of ratings. This algorithm enhances the precision of predictions when the data at hand is not dense enough. We evaluate the performance of different algorithms by running extensive experiments on the MovieLens [46] dataset. The algorithm based on random walks outperforms the simpler memory-based approaches.

Chapter 3 is devoted to the study of methods and algorithms of graph drawing. Graphs have various structures depending on the application they rise from. For example, Web graphs contain clusters of densely connected vertices with few connections between different clusters. On the contrary, engineering domains like 2/3 dimensional structural problems and electromagnetic applications generate graphs with very organized and symmetric structures. A good drawing is expected to reveal the structure of the graph. This objective is achieved by placing connected vertices closer to each other than non-connected vertices. The more the connected vertices get closer to each other than non-connected ones, the more the graph structure is abstracted. This means that empty areas get wider with respect to the surfaces occupied by the vertices. Depending on the structure of the graph, less or more abstraction may be needed to obtain a nice drawing. For example, in a graph with two clusters, we prefer that the two clusters be seen clearly apart from each other. Though, each cluster must not be overly concentrated as the details of the interactions within the clusters would not be seen. Most existing models draw graphs with a constant value of abstraction. We will discuss the problem of graph drawing in different levels of abstraction, and call it *flexible* graph drawing. Figure 1.3 clarifies the idea of flexible drawing. The figure shows a graph of 4 clusters in two different levels of abstraction. Each cluster contains 10 vertices. We first suggest the FlexGD, a force-directed model for flexible graph drawing. This model uses a mixture of linear and logarithmic functions as its attraction and repulsion terms of energy. The utility of linear and logarithmic force shapes in revealing the clusters has already been shown by Noack [101]. FlexGD upgrades this

property to adjustable cluster visualization, where the level of clustering can be adjusted. FlexGD delivers a unique representation of a graph where the vertices are spread uniformly over a disc. This leads to optimal utilization of the drawing surface. It is capable of drawing very weakly connected, even totally disconnected graphs. We will derive some provable properties of the minimum energy FlexGD layouts. After showing the properties of the model, a multilevel algorithm is suggested for finding the configuration with minimum energy. The main advantage of this algorithm over the previous ones is that it can cope with minimum energy states where the edge length distribution is not uniform. We also examine spectral approaches to graph drawing and mention how they can be modified in order to generate flexible drawings.

Another contribution of this chapter is to study drawing of *signed* graphs. In signed graphs, edges are labeled either as positive or as negative. They appear naturally in applications where the interaction between vertices happens to be repulsive. For instance, in social networks people can be grouped as friends and *foes*. Energy models are flexible enough to lend themselves to drawing signed graphs through integrating extra repulsive forces between negatively connected vertices. However, the shape of this force results in different behavior of the resulting model. For FlexGD this means different clusterings of the underlying signed graph. We will compare these results with a spectral approach applied to signed graphs by other authors.

The graph drawing approach discussed in Chapter 3 is one way to represent many types of data, including those issue of collaborative filtering. Namely, the collaborative filtering data can be modeled as a bipartite graph where the two vertex partitions model users and items. Edges would represent the ratings of users for items, with edge weights standing for the values of the ratings. This graph can then be visualized through graph drawing techniques. The drawback of this method is that placement of users and items in the space would not be a good indicative to predict the missing ratings. The reason is that drawing techniques are aimed at increasing the readability of the information rather than having satisfactory prediction capability. Collaborative filtering visualization can be done in a more sophisticated way: in Chapter 4 we suggest a unified approach based on a factorization-like scheme, generating 2-dimensional maps of collaborative filtering data which reflect the predictions of missing ratings. The core functionality of the suggested approach is a novel factorization-like algorithm projecting users and items in a high dimensional space such that the predicted rating of an item for a user is an inverse function of their Euclidean distance. The data is then projected into 2-dimensional space to be delivered to the end-user in the form of a 2-dimensional map. This type of presentation encapsulates more information than a top-k list, and its interpretation is quite intuitive. We will introduce two types of maps: Global Item Map (GIM) and Personalized Item Map (PIM). GIM represents the *whole* set of items such that similar items lie closer to each other. In PIM, one user is present together with a *limited* number of items, such that first, similar items lie close to each other. Second, the relevance of items to the active user decrease with their distance from her. The suggested method is not only a solution to visualization, but also a way to devise new evaluation methods for measuring the diversity of recommendation lists, and designing new hybrid recommender systems coping well with the multi-interest consuming habits of users. We conclude the document in Chapter 5 by summarizing the most important findings, and giving some directions for the future work.

CHAPTER 2

DECENTRALIZED RECOMMENDER SYSTEMS

Abstract

The need for efficient decentralized recommender systems has been appreciated for some time, both for the intrinsic advantages of decentralization and the necessity of integrating recommender systems into existing P2P applications. On the other hand, the accuracy of recommender systems is often hurt by data sparsity. In this chapter, we compare different decentralized user-based and item-based Collaborative Filtering (CF) algorithms with each other, and propose a decentralized user-based random walk approach, specifically designed to handle sparse data. We examine the performance of our approach in different settings by varying the sparsity, the similarity measure and the neighborhood size. We also introduce the *popularizing* disadvantage of the significance weighting term traditionally used to increase the precision of similarity measures, and elaborate how it can affect the performance of the random walk algorithm. The simulations on the MovieLens 10,000,000 ratings dataset demonstrate that over a wide range of sparsity, our algorithm outperforms other decentralized CF schemes.

2.1 Introduction

Different theoretical [33, 2, 84, 3] or empirical [56, 32, 88, 54, 76] approaches have addressed recommender systems. *Collaborative Filtering (CF)* is the most popular strategy in recommender systems. The reason behind this popularity is that CF requires no information about the content of the items, and has usually better precision than the other major approaches like the content-based approach. From among different strategies to CF, *matrix factorization* [76, 78] and *neighborhood-based models* [54, 115] are more popular than the others. Matrix factorization methods make users and items directly comparable by projecting them into the same latent feature space. The relevance of an item to a user is then inferred from user feedback and the feature space. Although matrix factorization slightly outperforms the neighborhood-based approach, this latter is the most widely used approach in CF due to some of its advantages like better explainability. In the neighborhood-based approach, explanations may be provided based on the similar users or items which have been central in the computation of the recommendations. On the contrary, explanations are hard to be given for the matrix factorization model, because the predictions are made by leveraging the whole information in the system, and not a restricted neighborhood. The matrix factorization approach is discussed in detail in Chapter 4. A neighborhood-based model consists of two phases: neighborhood formation and rating estimation. In the neighborhood formation phase, a set of similar items is formed for each item (item-based approach), or alternatively a set of similar users is formed for each user (user-based approach) based on a similarity measure. In the rating estimation phase, the neighborhood is used with a prediction function to estimate the ratings of items unknown to the users. Almost all the works in early days of CF were user-based algorithms. However, the item-based approach has recently received more attention in the domain of centralized recommender algorithms for its better scalability. More specifically, in most datasets, the number of users is larger and grows faster than the number of items. The item-based scheme also benefits from better explainability than the user-based scheme, because users have a better knowledge of items than of users.

Yet, recommender systems are confronted to a growing amount of data to process as the number of online users increases, and typically require expensive computational operations and significant storage to provide accurate results. While this combination of factors may saturate centralized systems, fully decentralized approaches provide an attractive alternative with multiple advantages. Firstly, the computation of the predictions can be distributed among all users, removing the need for a costly central server and enhancing scalability. Secondly, a decentralized recommender improves the *privacy* of the users for there is no central entity storing and owning the private information of the users. Several existing algorithms like [18], can be consequently deployed in decentralized environments to preserve the users' privacy by communicating users' opinions in encrypted form without disclosing their identity. Finally, a distributed recommender service is a valuable feature for peer-to-peer (P2P) applications like BitTorrent and Gnutella. These decentralized networks have become very popular media for users to share their content. They meet the very same problem as e-commerce websites: the large amount of available content makes it very difficult for a user to find items meeting her needs. Hence, P2P networks would greatly benefit from recommender systems.

Despite the numerous advantages that decentralized recommenders offer, the majority of existing work on recommendation algorithms has been focused on centralized systems so far. These algorithms are then not directly applicable to distributed settings. Developing recommender systems in decentralized contexts is more challenging than in centralized contexts. Namely, each user can only

leverage her own information and the data provided by a small (wrt the size of the system) number of other peers[1]. Hence, peers must have access to a decentralized mechanism efficiently detecting a limited number of similar users from among the whole set of peers in the system. We rely on epidemic algorithms as the decentralization method detecting a set of similar users.

The limitation of access to the data also accentuates the difficulties related to sparsity. As already mentioned, even in centralized contexts where the whole data is known, sparsity is one of the most important barriers in front of the recommender systems. For example, the MovieLens $10,000,000$ ratings dataset has a density of 1.31%. In a decentralized setting, only a subset of this already less data can be known by each peer. Therefore, the efficient use of the data at hand is even more essential than in centralized contexts.

In this chapter we suggest techniques for decentralization of neighborhood-based CF recommenders. CF is particularly suitable for the P2P context where no assumption can be made on the quality and the coherence of meta-data. As a consequence, content-based recommenders can not be easily applied. In addition, we suggest a model-based approach based on random walks to improve the precision of recommendations when the data is sparse. The details of the contributions of this chapter are as follows.

First, decentralized user-based and item-based CF algorithms are implemented in a P2P context using different similarity measures, and their performance is compared. We show that decentralized user-based approaches deliver good precision and less complexity than decentralized item-based approaches. In fact, decentralized user-based approaches do not suffer from drawbacks usually attributed to their centralized counterparts, specifically the lack of scalability.

Second, we propose a decentralized recommender system based on random walks. In our algorithm, each peer of the system is provided with a neighborhood composed of a small (wrt the size of the system) set of other similar peers by means of an epidemic protocol. Then, the ratings for unknown items of the neighborhood are estimated by running a random walk on this neighborhood. Once the peers have formed their neighborhood, i.e. the epidemic protocol has converged, each peer is thoroughly independent from other peers in generating her recommendations. This algorithm delivers better performance than previously considered decentralized CF algorithms when the data is sparse.

Third, the behavior of the random walk algorithm is discussed in detail in function of three parameters: sparsity, similarity measure and neighborhood size. This latter becomes specifically important in the P2P context, as it strongly affects the precision and the complexity of the algorithm. The parameter values for which our algorithm gives the best performance are empirically found for the MovieLens 10,000,000 ratings dataset. The random walk algorithm improves the precision over a wide range of sparsity while keeping the execution time affordable for peers.

2.2 Background

Recommender systems are modeled by a two-dimensional matrix denoted by R, with rows representing users and columns representing items. Each entry r_{ui} of R contains the rating of user u for item i. We assume a system with M users and N items, that is, $u \in \{1, 2, ..., M\}$ and $i \in \{1, 2, ..., N\}$. Each row R_{u*} is called the *rating vector of user* u, and each column R_{*i} the

[1]The terms peer and user are interchangeable in this chapter.

	i_1	i_2	i_3	i_3	i_5
u_1	r_{11}	?	?	?	r_{15}
u_2	?	?	?	r_{24}	?
u_3	?	?	r_{33}	?	r_{35}
u_4	r_{41}	?	?	?	?
u_5	?	?	r_{53}	?	?
u_6	r_{61}	?	?	?	?
u_7	?	r_{72}	?	?	r_{75}
u_8	r_{81}	r_{82}	?	r_{84}	?
u_9	?	?	?	r_{94}	r_{95}

Figure 2.1: An example of a rating matrix where missing ratings, represented by ?, are to be estimated.

rating vector of item i. The goal of the recommender system is to predict the missing entries of this matrix. An example of a rating matrix with 9 users, 5 items and 15 ratings is shown in Figure 2.1.

2.2.1 User-based Collaborative Filtering

User-based CF is presented in [54]. The idea behind the user-based collaborative filtering is that a user will be interested in the items which have received high ratings from users with similar taste to her. In this approach, the rating vectors of users, i.e. the rows of the rating matrix, are used to from a neighborhood of similar users around each user. Many similarity measures have been presented in the literature. One popular coefficient is the *Cosine similarity*. Cosine similarity between two users u and v is defined as:

$$cos(u,v) = \frac{\sum_{i \in I_u \cap I_v} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u \cap I_v} r_{ui}^2} \sqrt{\sum_{i \in I_u \cap I_v} r_{vi}^2}}, \tag{2.1}$$

where I_u and I_v are the set of items rated by u and v, respectively. A disadvantage of Cosine similarity is that it does not take into account the differences in users' rating behaviors. For example in a 5-star rating system, i.e. users rate the items from 1 to 5 stars, a user may rate from 3 to 5, but another one rates from 1 to 3 to reflect the same opinion on items. The *Pearson correlation* lifts this drawback by considering the offset of each rating from the user's mean rating. It is defined as:

$$\rho_{uv} = \frac{\sum_{i \in I_u \cap I_v} (r_{ui} - \overline{r}_u)(r_{vi} - \overline{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{ui} - \overline{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{vi} - \overline{r}_v)^2}} \tag{2.2}$$

where $\overline{r}_u$ is the mean rating of user u. Pearson correlation considers only the items rated by both users, but does not take into account the number of such items. As a result, a user may choose a neighbor with only very few items in common. To deal with this shortage, some authors opt for integrating a factor of *trust* into Pearson correlation known as *significance weighting* [54]. This is achieved by multiplying the Pearson correlation by a term reflecting the number of common items. For example in [54], this term is defined as $min(|I_v \cap I_u| / 50, 1)$. Once the neighborhood is

formed, the rating estimation phase is accomplished following a prediction rule, usually a weighted sum aggregation function:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N(u,i)} \omega_{uv}(r_{vi} - \bar{r}_v)}{\sum_{v \in N(u,i)} \omega_{uv}}, \tag{2.3}$$

where $\hat{r}_{ui}$ is the estimated rating of user u for item i, and $N(u,i)$ the set of k most similar users to u having rated i. Henceforth, we call ω_{uv} the *similarity weight* between users u and v. Depending on the setting, it can be either of the similarity measures presented in this section, or the output of the random walk algorithm.

2.2.2 Item-based Collaborative Filtering

The item-based approach is presented in [115]. It is assumed in the item-based collaborative filtering that users are interested in the items similar to those they have previously liked. The similarity between items is inferred from the ratings they have received from the other users. Namely, in contrast to the user-based approach, the item-based approach forms a neighborhood of similar items around each item. It operates on the columns of the rating matrix to compute the item-item similarities. The Cosine similarity between two items is defined as:

$$cos(i,j) = \frac{\sum_{u \in U_i \cap U_j} r_{ui} r_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} r_{ui}^2} \sqrt{\sum_{u \in U_i \cap U_j} r_{uj}^2}},$$

where U_i and U_j are the set of users having rated i and j, respectively. The item-based version of the Pearson correlation is defined as:

$$\rho_{ij} = \frac{\sum_{u \in U_i \cap U_j} (r_{ui} - \overline{r}_i)(r_{uj} - \overline{r}_j)}{\sqrt{\sum_{u \in U_i \cap U_j} (r_{ui} - \overline{r}_i)^2} \sqrt{\sum_{u \in U_i \cap U_j} (r_{uj} - \overline{r}_j)^2}},$$

where $\overline{r}_i$ is the average rating of item i. The users' different rating scales can be modeled by integrating the average rating of users into the definition of the Cosine similarity. This is known as the *adjusted Cosine*:

$$Acos(i,j) = \frac{\sum_{u \in U_i \cap U_j} (r_{ui} - \bar{r}_u)(r_{uj} - \bar{r}_u)}{\sqrt{\sum_{u \in U_i \cap U_j} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{u \in U_i \cap U_j} (r_{uj} - \bar{r}_u)^2}}.$$

In [115] the prediction is done using the weighted sum function:

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i,u)} \omega_{ij} * r_{uj}}{\sum_{j \in N(i,u)} \omega_{ij}}, \tag{2.4}$$

where $N(i,u)$ is the set of k most similar items to i that user u has rated. This can eventually be improved in order to capture the offset corresponding to different user attitudes:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{j \in N(i,u)} \omega_{ij}(r_{uj} - \bar{r}_u)}{\sum_{j \in N(i,u)} \omega_{ij}}. \tag{2.5}$$

ω_{ij} is the similarity weight between items i and j. In this chapter, it can be either of the similarity measures presented so far.

Algorithm 2.1 The skeleton of an epidemic algorithm executed by the peer u.

1: **loop**
2: wait T time units ▷ T is the gossip period.
3: $v \leftarrow selectDestination()$ ▷ destination selection.
4: $sendBuffer \leftarrow computeExchangeData()$ ▷ compute the data to be exchanged.
5: send $sendBuffer$ to v ▷ send the exchange data to the destination.
6: receive $receiveBuffer$ from v ▷ receive the data from the destination.
7: $view_u \leftarrow computeView()$ ▷ compute the new view.
8: **end loop**

2.2.3 Epidemic Protocols

In epidemic protocols (also known as *gossip protocols*) [126, 31, 11], each peer continuously exchanges information with other peers of the system. The goal is to search all the information in a decentralized way in order to find the information which is the most proper for the underlying application. Each peer is equipped with a cache containing a limited number of references to other users. The exchange of information is done through an active and a passive thread. There are three phases in each cycle of gossiping, described below.

- **Peer selection**: The source peer first needs to select a gossip target. The destination is chosen from the peers in the cache. There are different strategies for destination selection. A popular way of choosing the destination is to choose the oldest peer in the cache. This strategy allows for dynamic detection and elimination of the disconnected peers.
- **Data exchange**: Once a gossip destination is selected, the source decides upon what data it should exchange with the target. In the same way, the destination replies to the gossip request by sending back a fraction of its data. The exchanged data depends on the underlying application. Some application domains where gossiping can be applied include data dissemination, peer sampling, topology construction, resource management and computations [71].
- **Data processing**: upon the receipt of the gossip message, each peer compares the new data received from the other peer with its own data. If the new data contains elements more suitable for the underlying application, these elements replace the less proper part of the previous data.

Epidemic protocols are fully decentralized, can handle high churn rates, and do not require any specific protocol to recover from massive failures: they converge from any connected P2P network. Algorithm 2.1 shows the skeleton of an epidemic protocol.

2.2.4 Decentralized Recommenders

The need for effective decentralized recommender systems has been realized for some time. However, the research about them has remained modest with respect to the centralized algorithms. Notable works on the context are as follows. Tribler [123], a decentralized search engine using BitTorrent protocol, is capable of recognizing the user's taste and giving recommendations after a few search queries by the user. The user rating vectors are binary, i.e. the entry is 1, if the user has ever downloaded the corresponding item, and 0 otherwise. Tribler uses epidemic protocols to form the

neighborhood. Cosine similarity (Equation (2.1)) with significance weighting is used to compute the similarity between users. The significance weighting term is defined as ($|I_v| /40$), where v can be any of the neighbors. A non-normalized score is then computed for each item through the user-based CF approach. These scores are consequently used to generate an ordered recommendation list.

PocketLens [96] is a decentralized recommender algorithm developed by GroupeLens research group. In [96], different architectures from centralized to fully decentralized are suggested for neighborhood formation. PocketLens uses the Cosine similarity to estimate the similarity between users. Once a neighborhood of similar users is formed, an item-based algorithm is applied on the ratings existing in the neighborhood. The Cosine similarity is used as the similarity weight between items, and predictions are made using the weighted sum in Equation (2.4).

All of these works suggest *neighborhood-based* approaches to predict the missing ratings. On the contrary, we suggest a decentralized *model-based* approach based on random walks. The special asset of this algorithm is its higher precision in case of sparse data.

2.2.5 Solutions to Sparsity

Several solutions have been suggested to alleviate the problem of sparsity. Some works like [94, 22, 107] merge content-based recommenders with collaborative filtering to build hybrid recommenders which are more robust against sparse data. Gathering content data is challenging, specially in a P2P context where the user-generated tags are not coherent. Some authors exploit the demographic information [66, 82] of user's profiles like age, gender or code area to improve the recommendations when the data is not dense enough. As previously mentioned, such information is not easy to collect and endangers the privacy of the users. This matter is more restricting in P2P applications, because users are not required to register their information anywhere. Indeed, one of the motivations of the P2P users for employing these systems is to remain anonymous. *Default rating* [111] is another method for dealing with sparsity. This solution slightly improves the precision of the recommendations by assuming some default value for missing ratings. The disadvantage of this method is that it creates very dense input data matrix, hugely increasing the complexity of the computations. Hence, this is not an ideal solution for P2P, because the complexity of the algorithm must remain affordable for personal computers and laptops, being in general less powerful than central servers. Hence, a lot of effort has been made to develop models to mine further the existing data in order to detect potential hidden links between items or users. In [60] trust-based and item-based approaches are combined by means of a random walk model. The algorithm is centralized and the trust is explicitly expressed by the users. The information about trust does not exist in the majority of datasets including MovieLens. Authors in [134] suggest a random walk model in a centralized item-based CF approach. Their algorithm is to some extent similar to an item-based version of our random walk algorithm, but does not lend itself well to decentralized environments.

2.3 Decentralization of CF Algorithms

In this section, we describe the approach we have adopted to decentralize the neighborhood CF approaches. The main difficulty in decentralization of the neighborhood-based CF approach is the phase of neighborhood formation. In centralized recommender systems, the entire rating matrix R is input to the recommender algorithm. Consequently, the recommender algorithm can search

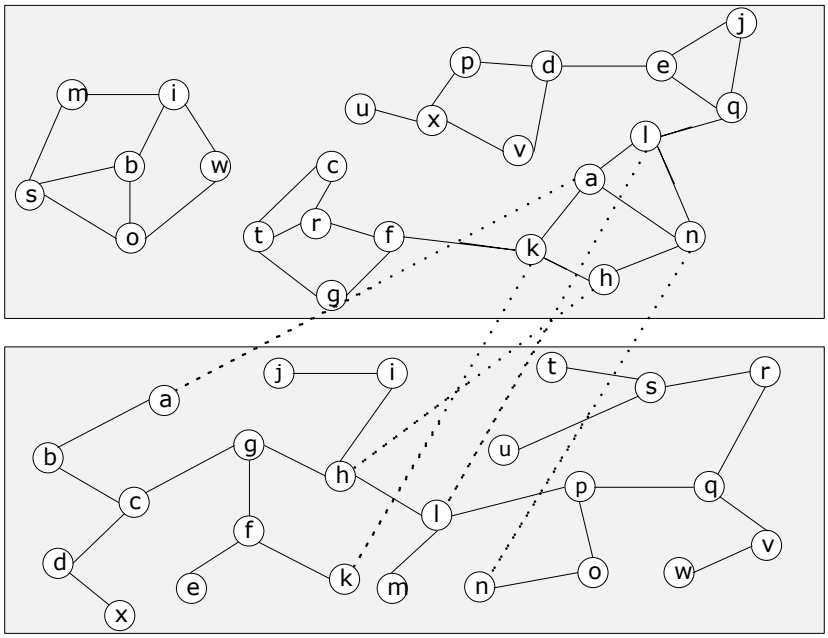


Figure 2.2: The association between the RPS and the clustering overlays of the P2P network.

among all the users or items to form a neighborhood for each user or item. On the contrary, in a decentralized system each user can only access to the data related to a limited number of other users. It is therefore critical to devise a protocol able to efficiently navigate through the P2P system and gather the most relevant data.

2.3.1 Decentralized User-based Approach

Epidemic protocols, described in Section 2.2.3, are very suitable for neighborhood formation in decentralized recommender algorithms. In order to converge fast to a view of similar users while ensuring the connectivity of the overlay network, peers run two gossip protocols: a Random Peer Sampling service (RPS) [64] and a *clustering protocol* [127]. The RPS provides each peer with a continuously changing random subset of the peers of the network. The clustering protocol provides each peer with a *view* of the network containing the information of the c peers maximizing a clustering function. The information of each peer is consisted of its reference and its rating vector. For our application, the clustering function reflects the similarity between the rating vectors of the source and the destination peers.

When a peer joins the network, her view is initialized at random through the RPS. The peer periodically selects a gossip target from her view and exchanges her view information with her. Upon reception of the new information, the peer computes her similarity with the new candidates and compares the results with her actual view and a set of random peers suggested by RPS. Then, keeping only the c most similar entries, she updates her view in order to improve its quality. While the clustering algorithm increases the risks of network partition, the RPS ensures connectivity with

high probability. Gossip clustering protocols are known for converging quickly to high quality views. By regularly communicating with the peers in the view, they also ensure their liveness and eliminate disconnected nodes. Figure 2.2 shows the association between the RPS and the clustering overlays.

We have tried Cosine, Pearson and Pearson correlation with significance weighting to form the neighborhood. Instead of following [54] to use $min(|I_v \cap I_u| / 50, 1)$ as the term of significance weighting, we have used the logarithm of the number of items both users have rated. The logarithmic function is damping. Hence, it is more discriminating for smaller numbers of common items. The reason of this choice is that choosing 50 as the minimum number of ratings not to be discriminated has been achieved empirically, and must be updated with the growth of the dataset and evolution of the user ratings. On the contrary, $\log$ copes better with the growth of the rating vectors. We call this the *modified Pearson correlation*:

$$corr(u, v) = \rho_{uv} \log(|I_v \cap I_u|). \tag{2.6}$$

It is shown in Section 2.5 that significance weighting tends to *popularize* the users' neighborhood, i.e. although adding this factor improves the total precision of the system, this leads to omitting users having few but valuable ratings, because the over-active (having rated a lot of items) users are favored over the users with more similarity but less items in common.

Only peers getting strictly positive similarity score are inserted in the view. When all the peers have selected their partial views, we define the *P2P network* (or the topology of it) as the network created by the peers connected via edges to the peers in their views. In a P2P system, for scalability reasons and network costs, c is typically small with respect to the size of the network. In order to obtain more data at a low network cost, each peer also uses the information of the peers in the view of her neighbors. Therefore, we define the *neighborhood* of each user as the peers directly connected and the peers connected within a distance of two hops in the P2P network. Depending on the clustering function, the size of the neighborhood can be up to $c^2 + c$. We evaluate the size of the neighborhood on the MovieLens dataset in Section 2.5.3. Figure 2.3 shows the neighborhood of user a on the overlay network.

Once the neighborhood is formed, the rating estimation of the decentralized user-based approach is done locally by each user in almost the same way it is done in its centralized counterpart. There is only a slight nuance explained here. In the centralized user-based approach, the neighborhood depends on the item for which the estimation is made. This is because if a fixed set of neighbors were chosen for all items, they would not have rated all the items for which we wanted to estimate the rating. In order to use the same number of users for each prediction, the neighborhood is chosen only from the users who have rated the corresponding item. Unlike in centralized systems, peers are not aware of all the existing items in a decentralized system. As a consequence, they do not know from the beginning which missing ratings are to be estimated. Furthermore, even if they can get access to this information through other procedures, it would be expensive from the network point of view to compute a neighborhood for each item. Hence, we form the neighborhood of each user only once through gossiping. Predictions are made for the items who are known by at least one of the neighbors. For each missing rating, all the existing ratings of the corresponding item in the neighborhood are used. Consequently, in contrast to the centralized version, a different number of ratings may be used for each prediction.

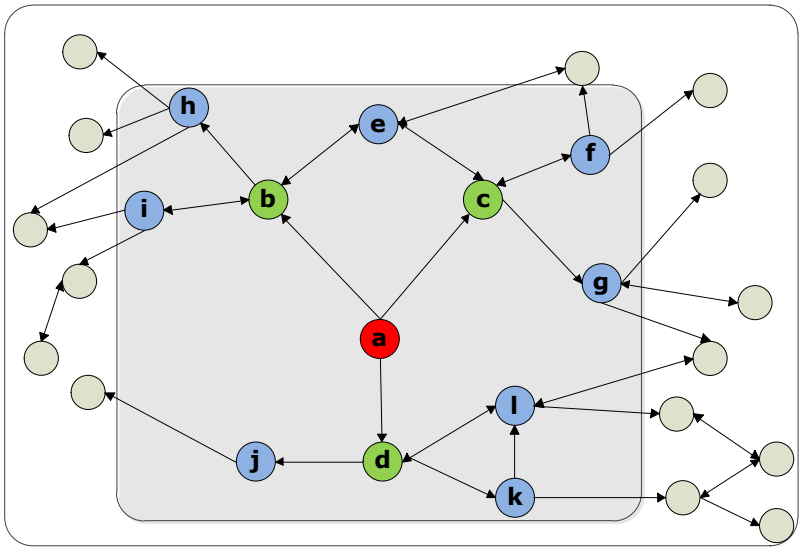


Figure 2.3: The neighborhood of user a on the P2P overlay network is consisted of users in 1 and 2 hop distance from her.

2.3.2 Decentralized item-based approach

The decentralization of the item-based CF algorithm is more of a challenge, because the algorithm needs the rating vectors of the items in order to find the similarity between them. This vector cannot be known by P2P users, as they do not know the ratings of the majority of the other peers. Consequently, similar to the user-based approach, each peer finds a neighborhood of similar users as a first step. A *partial* rating vector is then constructed for each item, based on the ratings available in the neighborhood. Predictions are made as they are made in the centralized item-based CF approach, using the similarity between the item whose rating is to be predicted and the items the user has rated. To compute the ratings, we tried Cosine, item-based Pearson and item-based Pearson with significance weighting as the similarity weight between items. We have used the logarithm of the number of common raters for two items as the term of the significance weighting. Hence, the item-based modified Pearson correlation is defined as:

$$corr(i,j) = \rho_{ij} \log(|U_i \cap U_j|). \tag{2.7}$$

2.3.3 Complexity of the Decentralized Neighborhood-based Approaches

The scalability advantage of the item-based CF over the user-based CF algorithms, holding in centralized settings, does not hold in P2P networks, because each user only computes her own recommendations. The complexity of CF algorithms is mostly due to the computation of similarity between users or items. For decentralized user-based CF algorithms, a vector containing the similarities between the central user and all the other peers in the neighborhood must be calculated. The complexity of each similarity calculation depends on the number of items in common between two

users. This number can go up to a considerable fraction of all items in the system. The complexity of the operation is then $O(SN)$ for each user, where S is the neighborhood size, and N the number of items in the whole P2P system. In decentralized item-based algorithms, the similarity between all unknown items of the neighborhood and the items the user has rated should be calculated to form a similarity matrix. We observed during simulations on the MovieLens dataset, that provided the neighborhood is big enough, the number of items in the neighborhood can get close to the total number of items in the system. This issue rises, because the neighborhood almost always contains some users having rated the majority of items. Hence, user u needs to compute $L(N - L)$ similarities, where L equals $|I_u|$. The complexity of each similarity calculation is related to the size of the rating vector of the items. It can be up to the neighborhood size. Hence, the complexity of the decentralized item-based approach is $O(N^2S)$ for each peer. The worst case happens when all items reside in the neighborhood, and the user has already rated half of them, that is, $L = N/2$. Comparing the complexity of the two approaches, it is clear that the decentralized user-based approach is much less complex for each peer than the decentralized item-based approach. This happens because in the decentralized user-based approach, each peer only computes the similarity between her and the users in the neighborhood, but does not need to compute the similarity between the other users. In the centralized version on the contrary, the complexity of the user-based approach is $O(M^2N)$, and the complexity of the item-based approach is $O(MN^2)$. Since the number of users is usually larger than the number of items, the user-based approach is more complex than the items-based approach. For the above reasons, the user-based approach seems to match better P2P settings. In Section 2.5, it is empirically shown that the decentralized user-based approach also has good precision.

2.4 Decentralized prediction through random walk

In this section, we present our decentralized model-based recommender algorithm. This algorithm applies a model based on random walks to have a more exact estimation of the similarity between users. The intuition behind this algorithm is to mine the neighborhood in longer paths to make more exact estimation of the similarity between users, instead of using only direct similarities, as in neighborhood-based approaches. Some centralized algorithms like [134] have used random walks to improve the accuracy of the predictions. In general, the recommendation problem is modeled by a weighted and directed graph where vertices represent the entity of interest. This entity are movies in an item-based movie recommender or webpages in the PageRank [102] algorithm for example. The application of random walks to centralized recommenders is relatively obvious. More precisely, the topology of the whole graph is known to the central recommender algorithm. Therefore, the algorithm can launch random walks from a vertex and output a similarity score for each of the other vertices. In other words, the random walk acts as a neighborhood formation mechanism on its own. In P2P however, this cannot be done, because the knowledge of each peer about the P2P network is limited to the peers of its neighborhood and the items they have rated.

In our decentralized algorithm, each peer first locally executes a neighborhood formation phase through gossip protocols. The users in the neighborhood are then modeled as a Markov chain graph, and a random walk is applied on this graph. Since the vertices represent the users of the neighborhood, we call our algorithm the *user-based random walk algorithm*. The neighborhood size will consequently be an important parameter of the algorithm, while in centralized algorithms it is always fixed to the size of the complete graph, i.e. the graph containing all users or items of the system. We will see in Section 2.5 that increasing the neighborhood size until some threshold

raises the precision of recommendations while keeping the execution time in a reasonable level. In the sequel, neighborhood formation and rating prediction based on random walks are discussed.

2.4.1 Neighborhood Formation

Peers rely on the clustering gossip protocol to hold a view of the c closest peers according to the similarity measure used for clustering. In the same way as discussed in Section 2.3, peers in one and two hop distance will be used to form a neighborhood of the size $O(c^2 + c)$.

2.4.2 Rating Estimation through the Random Walk Algorithm

In order to compute a personalized score prediction for an item, an active user a leverages all the scores that users in her neighborhood have assigned to that item. Each contribution is weighted to reflect the similarity between a and the corresponding user in the neighborhood. In traditional neighborhood-based recommender systems, the same similarity measure used for neighborhood formation (Pearson correlation or Cosine similarity for example) is used directly as the weight of each user's contribution. Instead, this weight is computed through the random walk algorithm in our approach. Each peer simulates the random walk on its neighborhood with a Markov chain. A Markov chain can be represented by a directed graph where the vertices are the states of the chain and the edges represent the transition probabilities from one state to another. In our case, the states symbolize the users in the neighborhood of a peer, let us say peer a. Let P^a be the transition probability matrix corresponding to the graph of user a's neighborhood. Each element p^a_{uv} of P^a represents the probability that u would ask v for recommendations. This probability is defined as the normalized similarity of the tail peer to the head. Another parameter $\beta \in (0, 1)$ is also added to the equation to consider the case where each peer jumps randomly to any other peer in the neighborhood during the random walk. Choosing very high values of β leads to assignment of equal transition probability towards all users in the neighborhood regardless of their similarity. It means that the ratings of all users will have the same weight in predictions. The value p^a_{uv} is computed using the following equation:

$$p^a_{uv} = (1 - \beta)\frac{s'_{uv,a}}{\sum_{z \in K(a)} s'_{uz,a}} + \frac{\beta}{m}, \tag{2.8}$$

where

$$s'_{uv,a} = \begin{cases} s_{uv} & \text{if } s_{uv} \geq 0 \text{ and } u \neq v \\ \gamma^a_u & \text{if } u = v \\ 0 & otherwise \end{cases},$$

$K(a)$ is the list of all users in the neighborhood of a and s_{uv} is the similarity between two users u and v. In the experiments presented in Section 2.5, s_{uv} is either the Pearson correlation or the Modified Pearson correlation. γ_u is the self loop parameter, modeling the case where a user answers the recommendation query before forwarding it to other users of the neighborhood. Since each user is logically more confident in her own opinion than that of any other user, we fixed γ^a_u as twice the similarity measure between u and the most similar user in her view:

$$\gamma^a_u = 2 \cdot \max_{v \in K(a), v \neq u} \{s_{uv}\}.$$

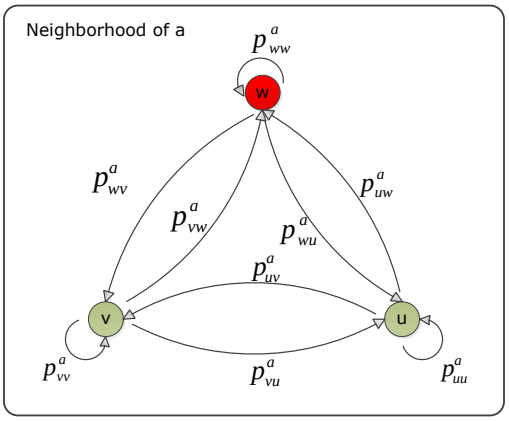


Figure 2.4: The Markov chain graph formed on the neighborhood of the active user a.

As a result of such modeling, a complete graph is formed from the users in the neighborhood of a, where the transition probability from vertex u to vertex v is computed through Equation (2.8). An an example of such a graph is shown in Figure 2.4 where the neighborhood of the active user a contains two other vertices u and v. The random walk starts from the users directly connected to the active user a, that is, peers having a one hop distance with the active peer in the network. The vector of initial probability distribution over the neighborhood is represented by $\vec{d}_a$. Each entry of $\vec{d}_a$ is defined as:

$$\vec{d}_a(v) = \frac{s''_{av}}{\sum_{z \in K(a)} s''_{az}}, \tag{2.9}$$

where

$$s''_{av} = \begin{cases} s_{av} & \text{if } v \in \text{clustering view of } a \\ 0 & otherwise \end{cases}.$$

Since each user computes her own predictions, we omit the index of a for the sake of simplicity. We use a finite length random walk where each peer decides to continue the walk with probability α. In Markov chains, the probability of being in a state at step k depends only on its previous state. Therefore, the probability of being in state u at step k is:

$$\Pr(X_k = u) = \alpha \sum_{v=1}^{m} \Pr(X_{k-1} = v) p_{vu} = \alpha^k \sum_{v=1}^{m} \vec{d}(v) P^k_{vu},$$

where m is the size of the active peer's neighborhood, and P^k the power k of the transition probability matrix. This is equal to the inner product of the initial distribution vector by column u of the P^k matrix:

$$\Pr(X_k = u) = \alpha^k (\vec{d} \cdot \vec{P}^k_{*u}).$$

The overall probability of being in state u is then:

$$\Pr(X = u) = \sum_{k=1}^{\infty} \alpha^k \vec{d} \cdot \vec{P}^k_{*u}.$$

At last, the final probability distribution vector over the neighborhood is:

$$\hat{R} = \sum_{k=1}^{\infty} \alpha^k \vec{d} P^k = \vec{d} \alpha P (I - \alpha P)^{-1}. \tag{2.10}$$

One way to obtain the final probability distribution vector is to launch several random walks on the graph, and take the average of all results. This is not needed however, and Equation (2.10) can be used directly to estimate the final distribution vector. Parameter α is optimized empirically. Once the final distribution vector is output by the random walk model, its entries are used as similarity weights ω_{uv} in Equation (2.3) in order to generate the recommendations.

2.4.3 Complexity of the User-based Random Walk Algorithm

The computation of the transition similarity matrix and the matrix inversion of Equation (2.10) are the main sources of complexity of the algorithm. The similarity must be calculated between each two users. The complexity of matrix inversion is $O(S^3)$, where S is the neighborhood size. Each similarity computation depends on the number of items in the neighborhood. If the set of items of the neighborhood gets close to the set of items in the whole system, the complexity of both operations becomes $O(S^2N + S^3)$. With a correct selection of neighborhood size, the algorithm gives very good performance with reasonable execution time.

In the same way, we can also imagine applying the same algorithm on the graph of items, then having an item-based random walk algorithm. The complexity of this algorithm will be $O(N^2S + N^3)$. Unfortunately, the execution time of item-based random walk algorithm is far from being affordable for the peers in real settings. The lack of efficiency of item-based random walk algorithm pushed us through suggesting the user-based random walk as a better approach for P2P applications.

2.5 Experiments and Results

In this section, we compare our user-based random walk algorithm with the decentralized neighborhood-based CF algorithms. The behavior of the random walk is also analyzed in Section 2.5.3.

2.5.1 Evaluation Methodology

In P2P systems the users do not report any feedback to a central server. As a result, we did not find any trace of real P2P rating data. In our experiments we use the MovieLens 10,000,000 ratings dataset [46]. This dataset was made available in three versions by GroupLens research group in January 2009 for research purposes about recommendations. The biggest version consists of $10,000,054$ ratings on $10,681$ movies, rated by $71,567$ real users of the MovieLens website, where each user has rated at least 20 movies. A 5-star scale is used to ask for ratings. To the best of our

knowledge, this is the second biggest dataset available after the Netflix dataset for research on recommender systems.

Since MovieLens is a central database, we adopt the following strategy to adjust it for our P2P experiments: For each user in the database, a peer object is instanced. This peer is attributed with the profile of the corresponding user in the database. This profile contains the list of films and corresponding ratings of the user. Consequently, each peer can access directly only to her own ratings, and needs to rely on the epidemic protocol described in Section 2.2.3 to find and retrieve the profiles of similar peers. This strategy enables us to simulate a P2P network composed of $71,567$ users residing in the MovieLens dataset, as if each of them had registered her ratings on her own computer instead of reporting them to the website.

We evaluate different recommender algorithms by cross validation. To generate proper training and test profiles, we split the profile of each MovieLens user into 20 regular random slices, where 20 comes from the minimum number of ratings per user in the MovieLens dataset. Consequently, each user has at least one rating in each of her profile slices. The training profile of a user contains a given number of slices and the remaining slices (out of 20) form the test profile. Since our algorithm targets situations where the input data is sparse, we tried different levels of sparsity by changing the proportion of the test and training profiles.

We use RMSE and Coverage to compare different algorithms with each other. Since predictions are computed locally for each user, the global definition of the RMSE and coverage, presented by Equation (1.2) and Equation (1.3), must be redefined per user. We define the RMSE of the predictions for user u as:

$$\mathrm{RMSE(u)} = \sqrt{\frac{\sum_{r_{ui} \in \mathrm{I_{T_u}}} (\hat{r}_{ui} - r_{ui})^2}{|\mathrm{I_{T_u}}|}}$$

Where $|\mathrm{I_{T_u}}|$ is the number of items in the test profile of u. The coverage of the recommender algorithm for user u is defined as the fraction of items in her test profile for which the algorithm can generate a prediction. It is defined as:

$$coverage(u) = \frac{\left|\hat{\mathrm{I}}_{\mathrm{T_u}}\right|}{|\mathrm{I_{T_u}}|},$$

where $\hat{\mathrm{I}}_{\mathrm{T_u}}$ is the set of predictable items in test profile of u. The total RMSE and coverage of the algorithm is then defined as the mean RMSE and the mean coverage of all peers, respectively.

2.5.2 Simulation Results

training set	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
α	0.7	0.9	0.8	0.7	0.8	0.6	0.5	0.5	0.5	0.3

Table 2.1: The value of optimal α for different levels of sparsity

The simulations are run for three view sizes: 10, 20 and 30. All results were obtained after 30 cycles of gossip, and the epidemic protocol had converged. β was fixed to 0.15 in the random walk algorithm. α was optimized by trying values in $(0, 1)$ with a step length of 0.1 in different levels of sparsity. The value of optimal α is given in Table 2.1. In general, the sparser the input data,

the larger α. This implies that the length of the random walk should increase as the data becomes sparser. The similarity between users is most often transitive in recommender systems. That is, if u is similar to v, and v is similar to z, u and z are similar with high probability. Despite, it happens in few cases that users in a two hop distance have negative similarity with the active user. We do not take such users into account when making predictions in user-based approaches, although they exist in the neighborhood. This issue never happens in the random walk algorithm, because the similarity weights generated by the algorithm are non-negative probabilities. In the same way, only items with positive similarity are used for prediction in item-based methods.

P-randomwalk	decentralized user-based random walk algorithm described in Section 2.4
MP-userbased	decentralized version of the user-based algorithm in [54] using Modified Pearson correlation
P-userbased	decentralized version of the user-based algorithm in [54] using Pearson correlation
Tribler[123]	decentralized user-based approach using Cosine with significance weighting
PocketLens [96]	decentralized item-based approach using Cosine
MP-itembased	decentralized version of the item-based algorithm in [115] using Modified Pearson correlation
P-itembased	decentralized version of the item-based algorithm in [115] using Pearson correlation

Table 2.2: Short description of decentralized CF algorithms with their abbreviations

We compare the random walk algorithm with 6 decentralized neighborhood-based algorithms. The main difference between these algorithms is their type (user-based or item-based), similarity measure used by the epidemic clustering protocol while forming the neighborhood and the similarity measure used for prediction. The description of these algorithms and the corresponding abbreviations are listed in Table 2.2. The best results, obtained with a view size of 30, is reported in Tables 2.3 and 2.4. The results for view sizes of 20 and 30 are summarized in Tables 2.5, 2.6, 2.7, and 2.8. The item scores computed by the original Tribler are binary. Instead, we generated a score for each item using Equation (2.3) to be able to compare it with the other algorithms. In P-itembased and MP-itembased, both neighborhood formation and item-based prediction are done using the same type of similarity measure. Namely, the neighborhood is formed using the (modified) user-based Pearson coefficient and the similarity weights between the items are computed using the (modified) item-based Pearson coefficient. Predictions are made using Equation (2.5).

As seen in Table 2.3, the P-randomwalk algorithm outperforms all other decentralized algorithms when the sparsity is less than 70%. P-userbased and MP-userbased approaches significantly outperform all the item-based approaches. Tribler shows the poorest performance among the user-based approaches, but still improves over the item-based approaches when sparsity is more than 5% and less than 25%. This shows that Pearson correlation is a better choice than Cosine similarity in the user-based approaches. PocketLens shows the best performance among the item-based approaches suggesting that Cosine similarity performs better than the item-based Pearson correlation in the item-based approaches. The Pearson correlation is centralized around the average rating. Hence, it takes into account the different rating habits of the users. The notion of rating habits is only associated with the users. On the contrary, the rating vector of an item is consisted of the ratings coming from different users. Consequently, centralizing the item rating vector about its mean is not as meaningful as it is for the user rating vector. This explains why Cosine similarity works better in the item-based approach, while Pearson correlation has better performance in the user-based approach.

Comparing MP-userbased and MP-itembased with P-userbased and P-itembased proves that significance weighting is efficient for both item-based and user-based approaches. All methods have

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	1.0719	1.0147	0.9869	0.9693	0.9575	0.9513	0.9423	0.9327	0.9196	0.8842
MP-userbased	1.1164	1.0481	1.0081	0.9841	0.9717	0.9662	0.9522	0.9408	0.9168	0.8752
P-userbased	1.1288	1.0594	1.0220	0.9980	0.9812	0.9725	0.9594	0.9477	0.9294	0.8903
Tribler	1.2301	1.0946	1.0439	1.0234	1.0166	1.0119	1.0050	0.9988	0.9892	0.9489
PocketLens	1.2036	1.1110	1.0595	1.0296	1.0119	0.9998	0.9833	0.9721	0.9553	0.9174
MP-itembased	1.2218	1.1410	1.0867	1.0493	1.0211	1.0011	0.9732	0.9559	0.9338	0.8985
P-itembased	1.2508	1.1601	1.0984	1.0524	1.0255	1.0062	0.9805	0.9656	0.9441	0.9038

Table 2.3: RMSE in different levels of sparsity, view = 30

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	0.8429	0.9272	0.9370	0.9487	0.9492	0.9506	0.9560	0.9540	0.9511	0.9474
MP-userbased	0.8324	0.9642	0.9854	0.9917	0.9943	0.9956	0.9969	0.9979	0.9983	0.9986
P-userbased	0.6971	0.8657	0.892	0.9220	0.9264	0.9316	0.9415	0.9407	0.9394	0.9364
Tribler	0.7469	0.9669	0.9881	0.9933	0.9952	0.9966	0.9978	0.9984	0.9990	0.9993
PocketLens	0.7435	0.9023	0.9337	0.9453	0.9515	0.9549	0.9583	0.9598	0.9582	0.9558
MP-itembased	0.8265	0.9612	0.9853	0.9924	0.9951	0.9963	0.9974	0.9979	0.9985	0.9989
P-itembased	0.6872	0.8844	0.9192	0.9406	0.9434	0.9470	0.9543	0.9521	0.9488	0.9458

Table 2.4: Coverage in different levels of sparsity, view = 30

good coverage when the training profile is more than 5%. However, the coverage of the methods using significance weighting, that is MP-userbased, MP-itembased and Tribler, is slightly better than other methods. The P-randomwalk algorithm improves the coverage over P-userbased although they use the same neighborhood. This is because P-randomwalk can also use the ratings of users having negative direct similarity.

In most recommender systems, the predicted scores are used to propose a recommendation list of top k items to the user. The quality of this list strongly depends on the RMSE of the system. The achievable RMSE lies in a very restricted range in available datasets, but it is proven that only slight improvement in RMSE yields much more satisfactory recommendation lists [76]. Hence, the improvement of our algorithm over the best of the previous algorithms is absolutely valuable.

The precision and coverage of all approaches increase with the size of the neighborhood. This improvement is due to the fact that algorithms rely on more users for making predictions. We observed during simulations that increasing the view size over 30 does not yield any significant improvement. Note that there is no advantage in choosing very large views. Not only does it increase the execution time, but also renders the recommendations less personalized. A view size about 30, allows for good precision and coverage while keeping the computation time quite affordable for the users. This value was computed for the MovieLens dataset, and may be different for other datasets.

2.5.3 Analysis of the Behavior of Random Walk

In this section we discover further how the behavior of the random walk changes with sparsity, neighborhood size and similarity measure.

Random Walk vs. Sparsity Random walk works well when the data is so sparse that classic similarity measures fail to detect meaningful relation between users. By increasing the training set proportion, classic similarity measures are able to deliver better performance than the random walk algorithm. For the view size of 30, P-randomwalk gives the best results until when the training set

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	1.1200	1.0519	1.0216	0.9988	0.9890	0.9806	0.9684	0.9608	0.9462	0.9097
MP-userbased	1.1567	1.0842	1.0368	1.0102	0.9957	0.9851	0.9706	0.9553	0.9313	0.8865
P-userbased	1.1670	1.1092	1.0679	1.0409	1.0243	1.0123	0.9953	0.9851	0.9659	0.9228
Tribler	1.2521	1.1362	1.0793	1.0563	1.0426	1.0392	1.0316	1.0249	1.0095	0.9658
PocketLens	1.2188	1.1455	1.0879	1.0518	1.0290	1.0133	0.9912	0.9786	0.9575	0.9174
MP-itembased	1.2375	1.1529	1.0991	1.0625	1.0352	1.0140	0.9852	0.9658	0.9401	0.9012
P-itembased	1.2658	1.1899	1.1281	1.0802	1.0495	1.0276	0.9969	0.9790	0.9538	0.9095

Table 2.5: RMSE in different levels of sparsity, view = 20

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	0.7454	0.8617	0.8771	0.8999	0.9004	0.9053	0.9157	0.9133	0.9072	0.9037
MP-userbased	0.7540	0.9449	0.9768	0.9857	0.9896	0.9919	0.9942	0.9955	0.9968	0.9975
P-userbased	0.5428	0.7543	0.8074	0.8512	0.8597	0.8708	0.8896	0.8899	0.8870	0.8857
Tribler	0.6662	0.9487	0.9811	0.9894	0.9927	0.9942	0.9963	0.9974	0.9985	0.9991
PocketLens	0.5401	0.7968	0.8623	0.8877	0.9006	0.9087	0.9169	0.9207	0.9194	0.9166
MP-itembased	0.7656	0.9478	0.9784	0.9876	0.9913	0.9933	0.9952	0.9963	0.9973	0.9979
P-itembased	0.5120	0.7897	0.8444	0.8833	0.8896	0.8983	0.9118	0.9096	0.9057	0.9016

Table 2.6: Coverage in different levels of sparsity, view = 20

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	1.2086	1.1473	1.1100	1.0826	1.0692	1.0587	1.0435	1.0337	1.0152	0.9686
MP-userbased	1.2160	1.1676	1.1108	1.0717	1.0516	1.0351	1.0127	0.9934	0.9643	0.9146
P-userbased	1.2035	1.1839	1.1529	1.1291	1.1111	1.0976	1.0800	1.0647	1.0423	0.9867
Tribler	1.2581	1.2076	1.1505	1.1128	1.0913	1.0763	1.0617	1.0507	1.0275	0.9790
PocketLens	1.2115	1.1957	1.1475	1.1065	1.0740	1.0507	1.0173	0.9972	0.9674	0.9166
MP-itembased	1.2564	1.1780	1.1197	1.0812	1.0549	1.0329	1.0037	0.9830	0.9545	0.9102
P-itembased	1.2538	1.2263	1.1783	1.1314	1.0982	1.0719	1.0335	1.0081	0.9710	0.9190

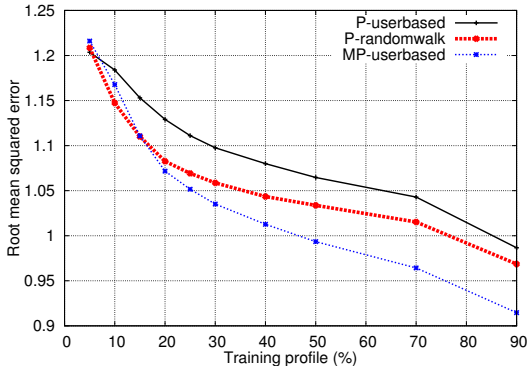
Table 2.7: RMSE in different levels of sparsity, view = 10

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	0.4312	0.6254	0.6681	0.7181	0.7273	0.7420	0.7717	0.7717	0.7676	0.7595
MP-userbased	0.4946	0.8470	0.9307	0.9557	0.9676	0.9747	0.9815	0.9851	0.9889	0.9911
P-userbased	0.2235	0.4392	0.5361	0.6156	0.6433	0.6698	0.7153	0.7221	0.7263	0.7227
Tribler	0.4587	0.8490	0.9344	0.9631	0.9748	0.9833	0.9901	0.9937	0.9970	0.9983
PocketLens	0.2126	0.4749	0.6115	0.6774	0.7152	0.7368	0.7641	0.7667	0.7830	0.7808
MP-itembased	0.5477	0.8808	0.9453	0.9645	0.9739	0.9793	0.9845	0.9874	0.9905	0.9921
P-itembased	0.2445	0.4985	0.6022	0.6795	0.7016	0.7251	0.7612	0.7664	0.7640	0.7596

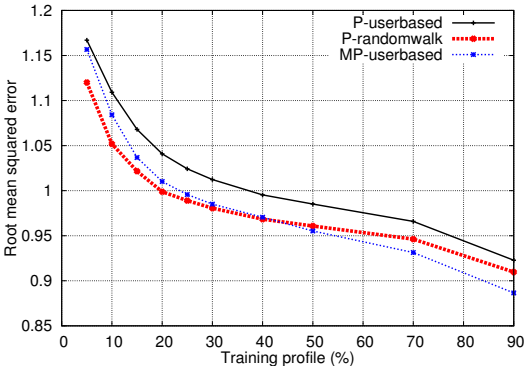
Table 2.8: Coverage in different levels of sparsity, view = 10

proportion is below 70%. However, when the training set proportion goes beyond 70%, the direct similarities become more reliable than the random walk.

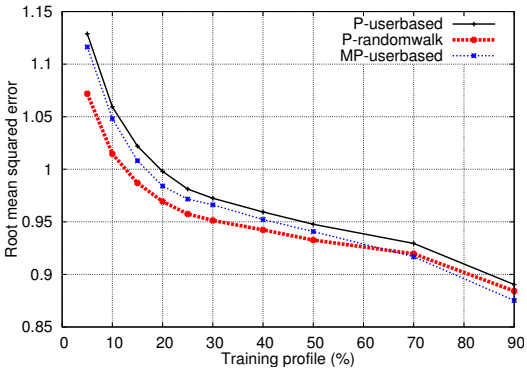
Random Walk vs. Neighborhood Size The precision of the three approaches with the best precision is plot in Figure 2.5. It is observed that before the training profile arrives at a threshold, P-randomwalk delivers the best performance in terms of precision by outperforming the MP-userbased algorithm as the second best approach. This threshold increases rapidly with an increment in the size of the neighborhood. It is 15% for a view size of 10, where the neighborhood is very small. It goes up to 40% for a view size of 20, and reaches 70% for the view size of 30, suggested as the best view size by our experiments. Furthermore, the amount of the improvement that P-randomwalk has over the other approaches increases with incrementing the neighborhood size. In fact, random walk re-evaluates the similarity weight between users by mining longer paths in the neighborhood to find



(a) view size = 10



(b) view size = 20



(c) view size = 30

Figure 2.5: RMSE

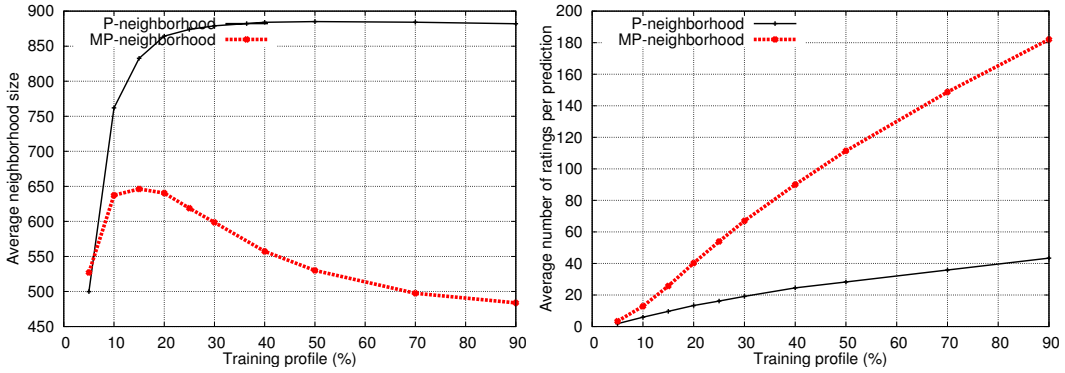
implicit transitive similarity between users. Classic similarity measures do not have such capability, since they can only capture direct relations. The chance for detecting such implicit relations is naturally higher for larger neighborhoods having more users. It is why P-randomwalk outperforms MP-userbased, but P-userbased has poorer precision than the latter, while both P-userbased and P-randomwalk use the same type of neighborhood.

Random Walk vs. Similarity Measure In this paragraph the effect of the similarity measure is studied by applying the random walk algorithm on two types of neighborhood formed through either Pearson correlation or Modified Pearson correlation.

To see how significance weighting of Modified Pearson correlation can influence the quality of the neighborhood, and in turn the performance of the random walk, we compared the average neighborhood size and the average number of ratings per prediction for the two types of neighborhood (see Figure 2.6).

It can be seen that MP-neighborhood has more ratings per prediction than P-neighborhood while its size is smaller. It means that Modified Pearson correlation prefers over-active users having in average rated a large number of items. Note that P-randomwalk has better precision than MP-userbased although it uses less ratings. It suggests that P-randomwalk *learns* much faster than MP-userbased. With increasing the training profile, the P-neighborhood approaches very soon its

maximum size, being about 900. Recall that the neighborhood contains peers within one and two hop distance on the P2P network. Unlike P-neighborhood, the size of MP-neighborhood decreases continuously when the training profile goes beyond 15%. This indicates that the P2P network becomes more clustered because the directly-connected peers will have many common neighbors in their views. In fact, with incrementing the training profile proportion, the chance of the over-active users for being put in the neighborhood becomes more than the moderate users, because such users have more ratings in each slice of their profile, and the significance weighting term grows faster for them. Consequently, the indegree of such users increases with a rise in the training profile. As a result, they will exist more or less in the neighborhood of all users. This is what we call the *popularizing effect* of significance weighting, briefly introduced in Section 2.2.1. Although the pop-



(a) view size = 30 (b) view size = 30

Figure 2.6: MP-neighborhood vs. P-neighborhood

ularizing effect leads to better coverage, it avoids the random walk algorithm from working in two ways. First, it decreases the chance of the random walk for precise re-evaluation of similarities by decreasing the neighborhood size and omitting users with few ratings but more *implicit* similarity to the central user. Second, the over-active users act as a *sink* in the Markov chain graph of the neighborhood during the random walk, because a lot of peers point towards them with large probability. Hence, the final probability of stopping on them at the end of the random walk is higher than other peers in the neighborhood. In other words, random walk intensifies the influence of the over-active users in making predictions with respect to the users with less ratings. This significantly decreases the quality of the random walk predictions in the MP-neighborhood. The neighborhood size of the MP-neighborhood has a peak when the training profile is 15%. This shows that the sinking behavior of significance weighting starts at this point. When the training profile is less than 15%, the P2P network is not still well clustered, and peers continue to add new users to their views.

To investigate the performance of the random walk algorithm on an MP-neighborhood, we implemented two new variants of our decentralized user-based random walk algorithm. The first variant is MP-randomwalk. Being quite similar to P-randomwalk, it uses Modified Pearson correlation instead of Pearson correlation for neighborhood formation and also as the user similarity weight s_{uv} in the Markov chain graph (see Equations (2.8) and (2.9)). The second one is MPP-randomwalk where the neighborhood is formed by Modified Pearson correlation, while the user similarity weights in the Markov chain graph are assigned using Pearson correlation. The precision of these approaches is compared with P-randomwalk in Table 2.9. The results are also plot in Figure 2.7.

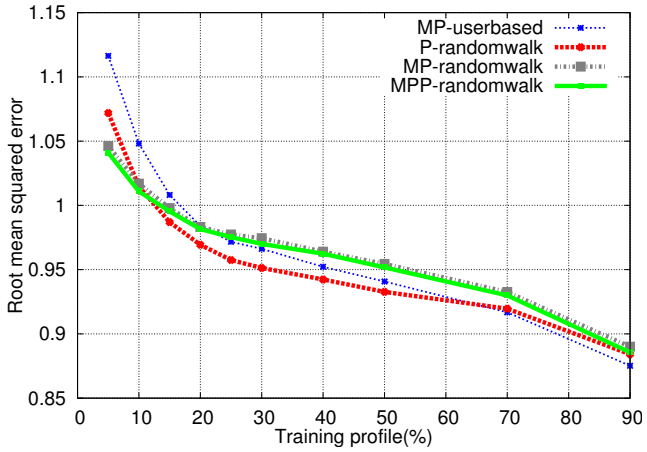


Figure 2.7: Error v/s proportion of data set used

It is seen in Figure 2.7 that when the training profile is more than 15%, MP-randomwalk starts to show much poorer results than P-randomwalk. Its performance is even worse than MP-userbased when the training set is more than 20%. The reason of this phenomenon hides behind the popularizing effect of significance weighting. The precision of MPP-randomwalk is slightly better than MP-randomwalk. This is due to the fact that the transition probability of the edges pointing towards the over-active users decreases when no significance weighting is used while assigning the user similarities. Hence, the sink role of such users is partly alleviated. It is also observed that MPP-randomwalk can outperform P-randomwalk when the training profile is extremely sparse (below 15%). This is due to the fact that the sinking behavior of Modified Pearson correlation is not still severe in this range.

2.6 Discussion

In this chapter, we propose a user-based random walk algorithm to enhance the precision of the decentralized CF algorithms. We use epidemic protocols to assign each user with a neighborhood of similar peers. Each user locally runs the random walk algorithm on her neighborhood, and computes her recommendations. The algorithm is fully decentralized, and the users are totally independent from each other in computing their own recommendations.

Decentralized item-based and user-based approaches were implemented using different similarity measures and compared with the user-based random walk algorithm. This latter showed the best precision over a wide range of sparsity. Decentralized user-based CF algorithms showed better precision and less complexity than their item-based counterparts. Moreover, Cosine similarity per-

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	1.0719	1.0147	0.9869	0.9693	0.9575	0.9513	0.9423	0.9327	0.9196	0.8842
MP-randomwalk	1.0462	1.0170	0.9979	0.9832	0.9773	0.9745	0.9640	0.9544	0.9326	0.8900
MPP-randomwalk	1.0410	1.0109	0.9953	0.9816	0.9753	0.9700	0.9624	0.9517	0.9299	0.8856

Table 2.9: RMSE of three variants of the random walk algorithm, view = 30

formed better in decentralized item-based algorithms, while Pearson correlation was a better choice for decentralized user-based algorithms.

Simulating a P2P network using the MovieLens 10,000,000 ratings dataset, we empirically showed how sparsity, neighborhood size, and similarity measure are determining parameters for the random walk algorithm. This algorithm delivers better precision when the data gets sparser. It works better for larger neighborhood sizes. The view size of 30 was given as a good trade-off between the precision and the execution time for MovieLens dataset. In the end, the behavior of the random walk was studied for two types of neighborhood formed either through Pearson correlation of Modified Pearson correlation. We showed how popularizing effect related to significance weighting term of Modified Pearson correlation is a barrier against the performance of the random walk algorithm.

CHAPTER 3

GRAPH DRAWING

Abstract

In this chapter we study force-directed algorithms and spectral approaches for graph drawing. We propose *FlexGD*, a novel force-directed algorithm for straight-line graph drawing. The algorithm strives to draw graph layouts encompassing from uniform vertex distribution to extreme structure abstraction. It is flexible for it is parameterized so that the emphasis can be put on either of the two drawing criteria. FlexGD is efficient for cluster visualization in an adjustable level. The energy function of FlexGD is minimized through a multilevel approach, particularly designed to work in contexts where the distribution of edge length over the layout is not uniform. Applying FlexGD on several real datasets, we illustrate both the good quality of the layouts on various topologies, and the ability of the algorithm to meet the addressed drawing criteria. Flexible drawing through the spectral approach is also discussed, and the layouts of the main drawing models are compared with each other. Finally, we discuss the drawing of signed graphs with force-directed and spectral approaches.

3.1 Introduction

Graph drawing is one of the branches of information visualization with applications in many domains including military, cartography, transportations, bioinformatics and social network analysis. The drawing of a graph called *layout* is the graphical representation of its vertices and edges in a 2 or 3 dimensional Euclidean space. We concentrate on *straight-line* graph drawing where the drawing problem is reduced to compute the coordinates of the vertices. Once these coordinates are found, edges are represented by straight lines between the corresponding vertices. Two prominent approaches to straight-line graph drawing are force-directed [40, 25, 68, 101, 36, 81] and spectral approaches [74, 19, 80]. Force-directed approaches associate the graph layout with a scalar value called the *energy* of the layout. The coordinates of the vertices in the optimal layout are found by minimizing the energy function. Spectral approaches define a *quadratic* energy function for the layout. The minimization problem of this function can be reformulated as a standard linear optimization problem. The solutions will be the (generalized) eigenvectors of a matrix related to the structure of the graph.

Graph drawing models reveal the underlying structure of graphs by placing the connected vertices closer to each other than the non-connected[1] vertices. In general, the level of structure abstraction is fixed, that is, the ratio of the total edge length to the total distance between all pairs of vertices is the same value for different executions of a model on the same graph. In this chapter, we suggest force-directed and spectral approaches for graph drawing with a varying level of structure abstraction. Namely, the user can control through a parameter *how much* the edges should be shorter than the non-edges.

We first suggest FlexGD, a Flexible force-directed algorithm for Graph Drawing. FlexGD draws graphs according to the two criteria of uniform vertex distribution and structure abstraction. The model is flexible, in that it is parameterized to be biasable towards any of the two drawing criteria, according to the user preferences. In general, the force or energy equations of the force-directed models are defined such that the resulting layouts meet some aesthetic criteria of drawing like uniform edge length and minimum edge crossing. Though, it has been shown in [101] that the beforementioned criteria prevent the conventional models from visualizing the clusters. Cluster visualization is of importance in applications like community detection in social networks. A new model called *LinLog* is then proposed by the author which is more proper for this goal. The energy function of FlexGD is closely related to that of LinLog, but is sophisticatedly modified to improve its features. Namely, the drawing area is filled optimally, and the layout is pleasing in the frontiers. Secondly, it upgrades the property of cluster visualization of LinLog to *abstractable* cluster visualization, that is, the user can decide upon the density of the clusters and to what extent they are set apart. As a consequence of this property, the graph structure can also be abstracted less or more. Finally, the model is capable of drawing disconnected graphs as most of the previous models have difficulties with their handling.

We derive formally some properties of FlexGD, and compare them with those of LinLog. The FlexGD energy function is minimized through a multilevel approach (discussed in Section 3.5.2). Existing multilevel algorithms work well if the edge length distribution is rather uniform. However, their performance on FlexGD is not convincing, as this latter enforces non-uniform distribution of edge length. The suggested multilevel algorithm for FlexGD can overcome the non-uniform dis-

[1] We call two vertices *non-connected*, if there is no direct edge between them. Two vertices are called *disconnected*, is there exists no path between them of the graph.

tribution of edge length. FlexGD layouts of some large real datasets are presented to illustrate the algorithm can generate quality layouts in a wide range of abstraction, satisfying different user preferences. We will also show how the spectral approach can be reformulated to generate flexible layouts. This spectral approach will not have the clustering properties of FlexGD, but its level of abstraction is adjustable. At the end, the generalization of force-directed and spectral approaches to *signed* graphs is discussed. In signed graphs edges are labeled as positive or negative, where positive edges denote attractive force and negative edges denote repulsive force between the connected vertices. Signed graphs appear in applications where the interactions within the data can be negative or positive. An example is social networks where people can be friends or foes.

3.2 Definitions and Notations

A d-dimensional layout $\mathbf{X}$ of a graph $G = (V, E)$ is a mapping of vertices into the Euclidean space $\mathbf{X} : V \longrightarrow R^d$, where every $i \in V$ is assigned with a coordinate vector x_i. This can be represented by an $n \times d$ matrix, where n is the number of vertices and d the number of dimensions. The ith row of $\mathbf{X}$ represented by x_i, is then the coordinates of vertex i. Each column of $\mathbf{X}$ represented by x^l, is the lth dimension of the layout. The Euclidean distance between i and j is denoted by $\|x_i - x_j\|$. Below, we define some notations from the literature of graph clustering, needed to express the properties of FlexGD and its signed generalizations. Also, some important results from linear optimization which are central in spectral graph drawing are presented.

3.2.1 Density and Distance

The *cut* and the *density* (also known as the *ratio cut*) are two measures widely used as the coupling between two sets of vertices. Minimizing the coupling is an established technique [89, 116] for finding the clusters. The cut between two disjoint sets of vertices V_1 and V_2 is defined as:

$$cut[V_1, V_2] = |E_{V_1 \times V_2}|,$$

where $E_{V_1 \times V_2}$ represents the set of edges between V_1 and V_2. The cut has the disadvantage of selecting biased clusters, i.e. one huge cluster against a tiny one. This is undesirable as clusters are supposed to contain a reasonably large group of vertices. One way to bypass this problem is to penalize small clusters by dividing the cut by the size of clusters. This leads to the definition of density:

$$density(V_1, V_2) = \frac{cut[V_1, V_2]}{|V_1|\,|V_2|}.$$

Arithmetic, geometric and harmonic mean distance are the most popular definitions in the literature to measure the mean distance between a set of vertices on a layout. For $F \subset V^{(2)}$ and layout $\mathbf{X}$, they are defined as:

$$arith(F, \mathbf{X}) = \frac{1}{|F|} \sum_{\{i,j\} \in F} \|x_i - x_j\|.$$

$$geo(F, \mathbf{X}) = \sqrt[|F|]{\prod_{\{i,j\} \in F} \|x_i - x_j\|}.$$

$$harm(F, \mathbf{X}) = \frac{|F|}{\sum_{\{i,j\}\in F} \frac{1}{\|x_i - x_j\|}}.$$

Arithmetic mean is a linear averaging operator. On the contrary, geometric mean and harmonic mean are non-linear operators with different properties. The geometric mean reinforces the effect of very small distances while attenuating the effect of very large ones. The harmonic mean shows sharp fluctuations, provided a large number of pairwise distances decrease or increase at the same time. In the theorems about the properties of FlexGD, and the generalizations about signed graphs, the abovementioned definitions will appear naturally in slightly modified forms which are intuitively easy to understand. As a consequence, we found it helpful to generalize these definitions in order to posit the theorems in a more convenient form. For $F \subset V^{(2)}$ the *generalized arithmetic mean* is defined as:

$$genarith(F, \mathbf{X}, k) \quad = \quad \frac{\sum_{\{i,j\}\in F} \lambda_{ij}\|x_i - x_j\|}{\sum_{\{i,j\}\in F} \lambda_{ij}}, \quad \lambda_{ij} = \begin{cases} k+1 & \text{if } \{i,j\} \in E_F \\ 1 & \text{otherwise} \end{cases},$$

where E_f is the set of edges over F. This definition counts the connected pairs $k+1$ times the non-connected pairs. Consequently, shortening the distance between connected vertices causes more decrease in $genarith(F, \mathbf{X}, k)$ than shortening the distance between non-connected vertices.

3.2.2 Signed Graphs

We represent signed graphs with $G_s = (V, E)$, where $E = E^+ \cup E^-,\ E^+ \cap E^- = \emptyset$. E^+ and E^- denote the sets of positive and negative edges, respectively. The positive and negative cut between two disjoint clusters $V_1,\ V_2 \subset V$ are defined as:

$$cut^+[V_1, V_2] = \left|E^+_{V_1\times V_2}\right|, \quad cut^-[V_1, V_2] = \left|E^-_{V_1\times V_2}\right|,$$

where $E^+_{V_1\times V_2}$ and $E^-_{V_1\times V_2}$ indicate the set of positive and negative edges between V_1 and V_2, respectively. Then, *positive density* and *negative density* between V_1 and V_2 are defined as:

$$density^+(V_1, V_2) = \frac{cut^+[V_1, V_2]}{|V_1|\,|V_2|}, \qquad density^-(V_1, V_2) = \frac{cut^-[V_1, V_2]}{|V_1|\,|V_2|}.$$

The *positive generalized arithmetic mean distance* for $F \subset V^{(2)}$ on layout $\mathbf{X}$ is defined as:

$$genarith^+(F, \mathbf{X}, k) \quad = \quad \frac{\sum_{\{i,j\}\in F} \lambda_{ij}\|x_i - x_j\|}{\sum_{\{i,j\}\in F} \lambda_{ij}}, \quad \lambda_{ij} = \begin{cases} k+1 & \text{if } \{i,j\} \in E^+_F \\ 1 & \text{otherwise} \end{cases},$$

where E^+_F is the set of positive edges in F. Note that $genarith^+(F, \mathbf{X}, k)$ has the same definition as the generalized arithmetic mean, but only takes into account the positive edges of the graph. In the same way the *negative generalized geometric mean distance* is defined as:

$$gengeo^-(F, \mathbf{X}, k) \quad = \quad \sqrt[(\sum_{\{i,j\}\in F} \lambda'_{ij})]{\prod_{\{i,j\}\in F} \|x_i - x_j\|^{\lambda'_{ij}}}, \quad \lambda'_{ij} = \begin{cases} k+1 & \text{if } \{i,j\} \in E^-_F \\ 1 & \text{otherwise} \end{cases},$$

where E_F^- is the set of negative edges in F. The *negative generalized harmonic mean distance* of F on layout $\mathbf{X}$ is defined as:

$$genharm^-(F, \mathbf{X}, k) = \frac{\sum_{\{i,j\}\in F} \lambda'_{ij}}{\sum_{\{i,j\}\in F} \frac{\lambda'_{ij}}{\|x_i - x_j\|}}, \quad \lambda'_{ij} = \begin{cases} k+1 & \text{if } \{i,j\} \in E_F^- \\ 1 & \text{otherwise} \end{cases}.$$

The *positive-negative generalized arithmetic mean distance* of F on layout $\mathbf{X}$ is defined as:

$$genarith^{\pm}(F, \mathbf{X}, k_1, k_2) = \frac{\sum_{\{i,j\}\in F} \lambda''_{ij}\|x_i - x_j\|}{\sum_{\{i,j\}\in F} \lambda''_{ij}}, \quad \lambda''_{ij} = \begin{cases} k_1+1 & \text{if } \{i,j\} \in E_F^+ \\ -k_2+1 & \text{if } \{i,j\} \in E_F^- \\ 1 & \text{otherwise} \end{cases}.$$

We assume that the relation between $\|x_i - x_j\|$, k_1 and k_2 is such that $genarith^{\pm}(F, \mathbf{X}, k_1, k_2)$ results in a positive value. We will see in Section 3.8 that this assumption is satisfied for the special case we are interested in. In the same way, more generalizations like $genarith^-(F, p, k)$, $gengeo^+(F, p, k)$ and $genharm^+(F, p, k)$ may also be defined. Though, they will not appear in our theorems.

3.2.3 Spectral Graph Theory

The most important matrices used in the spectral graph theory are the *Adjacency* and the *Laplacian* matrices. The Adjacency matrix of a graph, represented by $A \in \mathbb{R}^{n\times n}$, is a symmetric matrix where the entries show the edge weights of the graph:

$$A_{ij} = \begin{cases} \omega_{ij} & i \neq j \\ 0 & i = j \end{cases}, \tag{3.1}$$

where ω_{ij} is the weight of the edge between i and j. It is zero if there is no edge between i and j. The *Degrees matrix* of a graph, represented by $D \in \mathbb{R}^{n\times n}$, is a diagonal matrix where each non-zero entry represents the *degree* of the corresponding vertex, that is, $D_{ii} = \sum_{k\neq i} \omega_{ik}$. The Laplacian of a graph denoted by $L \in \mathbb{R}^{n\times n}$ is defined as $L = D - A$:

$$L_{ij} = \begin{cases} -\omega_{ij} & i \neq j \\ D_{ii} & i = j \end{cases}. \tag{3.2}$$

The most important properties of the Laplacian are as follows.

1. It is symmetric and positive semi-definite, that is, $\forall x \in \mathbb{R}^n : x^T L x \geq 0$.
2. Its smallest eigenvalue is 0, corresponding to the eigenvector $\mathbf{1}_n$.
3. All other eigenvalues of L are non-negative and real-valued.
4. For any column vector $x \in \mathbb{R}^n$:

$$x^T L x = \frac{1}{2} \sum_{\{i,j\}\in E} \omega_{ij}(x(i) - x(j))^2. \tag{3.3}$$

Proof

$$x^T Lx = x^T(D-A)x = x^T Dx - x^T Ax = \sum_{i=1}^{n} D_{ii}x(i)^2 - \sum_{i,j=1}^{n} \omega_{ij}x(i)x(j) =$$
$$\frac{1}{2}(\sum_{i=1}^{n} D_{ii}x(i)^2 - 2\sum_{i,j=1}^{n} \omega_{ij}x(i)x(j) + \sum_{j=1}^{n} D_{jj}x(j)^2) =$$
$$\frac{1}{2}\sum_{i,j=1}^{n} \omega_{ij}(x(i)-x(j))^2 = \frac{1}{2}\sum_{\{i,j\}\in E} \omega_{ij}(x(i)-x(j))^2.$$

The spectral graph drawing approaches are notably based on the following two important results from the theory of linear optimization. The proofs can be found in standard text books of linear algebra like [43, 21] or in [74]. Here, they are stated as in [74]. Theorem 3.2.1 is used for graph drawing using the eigenvectors of the Laplacian.

Theorem 3.2.1 *Given a symmetric matrix $A_{n\times n}$, denote by $u_1, \ldots, u_n$ its eigenvectors, with corresponding eigenvalues $\mu_1 \leq \ldots \leq \mu_n$. Then, $u_1, \ldots, u_p$ are an optimal solution of the constrained minimization problem:*

$$\min_{x^1,\ldots,x^p} \sum_{k=1}^{p} (x^k)^T Ax^k$$
$$\text{subject to :} (x^k)^T x^l = \delta_{kl} \quad k,l = 1,\ldots,p,$$

where

$$\delta_{kl} = \begin{cases} 1 & \text{if} \quad k = l \\ 0 & \text{otherwise} \end{cases}.$$

For graph drawing using the degree-normalized eigenvectors, a more general form of Theorem 3.2.1 is applied:

Theorem 3.2.2 *Given a symmetric matrix $A_{n\times n}$ and a positive definite matrix B, denote by $u_1, \ldots, u_n$ the generalized eigenvectors of (A, B), with corresponding eigenvalues $\mu_1 \leq \ldots \leq \mu_n$. Then, $u_1, \ldots, u_p$ are an optimal solution of the constrained minimization problem:*

$$\min_{x^1,\ldots,x^p} \sum_{k=1}^{p} (x^k)^T Ax^k$$
$$\text{subject to :} (x^k)^T Bx^l = \delta_{kl} \quad k,l = 1,\ldots,p.$$

3.3 Background

In this section we briefly introduce the most important works on graph drawing.

3.3.1 Force-directed Approach

Force-directed algorithms [40, 25, 68, 101, 36, 81] are the most popular approach applied to graph drawing in practice. Their popularity rises from high layout quality, ease of implementation, code flexibility and intuitive understanding. Force-directed algorithms do not need any initial knowledge about the topology of the graph. They model vertices as a collection of particles and assign them attractive and repulsive forces according to force shapes improvised from physical metaphors like springs and electrical charges. The algorithm lays out the graph from an initial random configuration computing the net force on each vertex and moving the vertices iteratively until an equilibrium state is achieved between all forces. Force-directed algorithms are composed of two components: the energy function and the minimization algorithm. The energy function assigns a scalar energy value to the layout. Attractive and repulsive forces are linked to the energy function, as force is the minus gradient of energy. The role of the minimization algorithm is to compute a force equilibrium in the system, being equivalent to a local minimum of the energy function.

Besides their advantages, there are a number of drawbacks associated with force-directed approaches. The most important of them is the difficulty of finding the global minimum of the energy function. In general, minimizing the energy function is NP-hard. Hence, minimization algorithms are heuristics finding a local minimum of the function. Force-directed approaches also suffer from the lack of strong theoretical support about the properties of the resulting layouts. Force-directed layouts depend on the initial configuration, as the minimization heuristic may rest in different minima for different executions of the algorithm. Consequently, the reliability of the layout depends strongly of the topology of the underlying graph. For some topologies, it is challenging to generate a nice layout. Force-directed algorithms are characterized with high running time. Although with recent advances, specially the advent of multilevel approaches [128, 51, 57, 80], they are more efficient and faster, their running time is still high comparing to the spectral approach. Despite their shortcomings, force-directed algorithms have remained the dominant approach to graph drawing, because their layouts are usually more pleasing than the other approaches. The main force-directed models are listed below.

Spring Embedder

The *Spring Embedder* method [36] by Eades was one of the earliest variants of the force-directed algorithms. This model considers a graph as a collection of rings connected through springs. However, Eades replaces the Hook's law by his own force equations. Furthermore, in contrast to a real mechanical system of rings and springs, in the model of Spring Embedder all vertices repulse each other, but only connected vertices attract each other. The attraction and repulsion forces exerted on vertex i from vertex j in the Spring Embedder model are defined as:

$$\begin{aligned} \vec{f_a}(i,j) &= k_1 \log \frac{\|x_i - x_j\|}{k_2}(x_j - x_i) \text{ for } \forall\{i,j\} \in E \\ \vec{f_r}(i,j) &= -\frac{k_3}{\|x_i - x_j\|^2}(x_j - x_i) \text{ for } \forall\{i,j\} \in V^{(2)}, \end{aligned}$$

where k_1, k_2 and k_3 are constants. In order to find the minimum energy configuration, Eades computes the forces between all pairs of vertices per cycle, and displaces them with a step length until the algorithm converges. This suggests a complexity of $O(|V|^2+|E|)$ per iteration. In practice, the Spring Embedder algorithm is not scalable to large graphs we encounter.

Spring Electrical Model

Fruchterman and Reingold [40] suggest the Spring Electrical model for graph drawing. This model considers a graph as a collection of electrically charged rings where connected vertices are linked through springs. Therefore, all vertices repulse each other, while the attraction exists uniquely between connected vertices. The authors present the forces exerted on vertex i from vertex j as:

$$\begin{aligned} \vec{f_a}(i,j) &= \frac{(\|x_i - x_j\|)^2}{k}(x_j - x_i) \text{ for } \forall\{i,j\} \in E \\ \vec{f_r}(i,j) &= -\frac{k^2}{(\|x_i - x_j\|)}(x_j - x_i) \text{ for } \forall\{i,j\} \in V^{(2)}, \end{aligned}$$

where k is a constant. It is proved that the value of k has no effect on the resulting layout but scaling. The minimum energy layout is computed through an iterative algorithm. However, in order to decrease the complexity of the computations, only the repulsive forces between the vertices in a local neighborhood are computed, instead of computing the repulsive forces between all vertices. Namely, the rectangular drawing area is divided into a grid box, and the repulsive forces exerted on each vertex are only computed for the vertices in its area and those in the neighboring areas. The best case running-time of this algorithm is $O(|V| + |E|)$, but the worst case running time remains $O(|V|^2 + |E|)$. Although this approach decreases the complexity, it can degrade the quality of drawings, as a local computation of the repulsive forces is not always sufficient for distribution of far away vertices.

Davidson and Harel Model

Davidson and Harel [25] suggest the energy-based version of their model integrating terms addressing uniform vertex distribution, edge shortening and fitness within drawing boundaries. The authors mention these criteria are enough to generate nice layouts of most graphs. They also argue that as a result of the attraction forces between edges and the pairwise repulsion between vertices, issues related to edge crossing and edge-vertex crossing are already handled to some good extent. Nevertheless, they suggest integrating extra terms into the energy function to emphasize explicitly the minimization of edge and edge-vertex crossings. The authors use a technique called simulated annealing to minimize their cost function. This method is very computationally expensive, and is not effective for graphs with more than a few tens of vertices. Being complex as $O(|E| \cdot |V|^2)$, it is no more of practical interest in graph drawing, as more advanced and elaborate techniques have been invented.

In our implementation, we only consider the first two criteria. Considering boundary fitting is not needed, as the layout can be scaled to fit within the valid output boundary. The derivation of Davidson and Harel energy function that we implemented is:

$$DH(\mathbf{X}) = \sum_{\{i,j\} \in E} \|x_i - x_j\|^2 - \sum_{\{i,j\} \in V^{(2)}} \frac{1}{\|x_i - x_j\|^2}.$$

Instead of simulated annealing, we use an iterative gradient descent algorithm, mostly similar to works like [40, 57], to minimize Davidson and Harel energy function.

Multilevel Algorithms

All of the force-directed algorithms presented so far suffer from the lengthy time of computations. The Barnes and Hut algorithm [5] is a way to alleviate this problem by decreasing the complexity of computing the pairwise repulsive forces to near linear time. The algorithm was originally proposed for the problem of N-body simulations in physics. It was initiated to graph drawing by Tunkelang [125]. The idea is to decrease the complexity of computing the pairwise repulsive forces by approximating the nearby vertices as a single vertex, if they are sufficiently far away from the active vertex. This algorithm decreases the complexity of the algorithm to $O(|V|\log|V|+|E|)$.

The Barnes and Hut algorithm can speed up the computation of repulsive forces, but the problem of escaping the local minima of the energy function still persists. In the beginning of 2000 years, many works like [47, 129, 57, 109] adopted a new approach called the *multilevel* approach to improve the quality of drawing by more efficient minimization of the energy function. In multilevel approaches, the graph is *coarsened* to a series of smaller graphs such that each coarser graph reflects the structure of the finer one. We postpone the detailed discussion of the Barnes and Hut algorithm and the multilevel scheme to Section 3.5, as derivations of these methods are used as fundamental blocks of our minimization algorithm.

Stress Model

The Stress model [68] by Kamada and Kawai relates the drawing to the isometry w.r.t. the graph theoretic distance. The model assigns a spring between each two vertices. Hence, vertices repulse each other, if they get closer than their ideal distance, and attract each other, if they get farther than that. The energy function of the Stress model is defined as:

$$Stress(\mathbf{X}) = \sum_{i<j} \omega_{ij}(\|x_i - x_j\| - d_{ij})^2,$$

where d_{ij} is the ideal distance between i and j, and ω_{ij} a weighting factor chosen as $\omega_{ij} = \frac{1}{d_{ij}^2}$. d_{ij} is usually set to the graph-theoretic distance between i and j. The Stress energy function can be minimized using the gradient descent approach. In this case the force exerted on vertex i from vertex j is:

$$\vec{f}(i,j) = \omega_{ij}(\|x_i - x_j\| - d_{ij})\frac{(x_j - x_i)}{\|x_j - x_i\|}.$$

Notice that both attractive and repulsive forces follow the same equation. Nevertheless, a significantly better approach to minimize the Stress function is the *majorization* technique. The superiority of this method over the gradient descent is in terms of better guarantee of convergence towards the global minimum and lower execution time. This technique was originally proposed for Multi Dimensional Scaling (MDS) in [28]. Derivation of majorization was applied to graph drawing by Gansner et al. in [41]. In majorization technique, the Stress function is bounded from above by a convex function called the *majoring* function. The majoring function is such that it coincides with the Stress function in a *supporting point*. It can be proved that minimization of the Stress function is equivalent to iteratively minimizing the majoring function. The advantage is that the majoring function can be minimized relatively easily. In order to bound the stress function, let us rewrite the stress function in the form of:

$$Stress(\mathbf{X}) = \sum_{i<j} \omega_{ij}\|x_i - x_j\|^2 + \sum_{i<j} \omega_{ij} d_{ij}^2 - \sum_{i<j} 2\|x_i - x_j\|\omega_{ij} d_{ij}.$$

The second term is a constant. The first term is in quadratic form and can be written in terms of the Laplacian of the graph. Observe that $\sum_{i<j} \omega_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|^2 = \mathrm{Tr}(\mathbf{X}^T L \mathbf{X})$. The last term may be bounded with a quadratic function using the Cauchy-Schwartz inequality. Namely, for any $\mathbf{Y} \in \mathbb{R}^{n \times d}$: $\|x_i - x_j\| \|y_i - y_j\| \geq (x_i - x_j)^T (y_i - y_j)$. The third term of the Stress function is then bounded by:

$$\sum_{i<j} \|x_i - x_j\| \omega_{ij} d_{ij} \geq \sum_{i<j} \frac{(x_i - x_j)^T (y_i - y_j)}{\|y_i - y_j\|} \omega_{ij} d_{ij}.$$

This inequality can be written in the matricial form:

$$\sum_{i<j} \|x_i - x_j\| \omega_{ij} d_{ij} \geq \mathrm{Tr}(\mathbf{X}^T L^Y \mathbf{Y}),$$

where

$$L^Y_{ij} = \begin{cases} -\omega_{ij} d_{ij} \|y_i - y_j\| & i \neq j \\ -\sum_{i \neq k} L^Y_{ik} & i = j \end{cases}.$$

The Stress function is then bounded as:

$$Stress(\mathbf{X}) \leq \sum_{i<j} \omega_{ij} d_{ij}^2 + \mathrm{Tr}(\mathbf{X}^T L \mathbf{X}) - 2\mathrm{Tr}(\mathbf{X}^T L^Y \mathbf{Y}).$$

The equality is held if $\mathbf{X} = \mathbf{Y}$. Differentiating by $\mathbf{X}$, we obtain:

$$L\mathbf{X} = L^Y \mathbf{Y}.$$

The answer to this equation can be found by iteratively solving $L\mathbf{X}(t+1) = L^{\mathbf{X}(t)}\mathbf{X}(t)$. This latter can be solved through methods like conjugate gradient or Lanczos algorithm.

Binary Stress Model

The disadvantage of the Stress model is that the graph theoretic distances must be computed in advance for all pairs of vertices. This increases the complexity of the computations. Koren and Çivril [81] propose the Binary Stress model bridging the Spring Electrical model to the Stress model. In this model the vertex overlap is avoided by introducing a uniform pairwise distance for non-connected vertices. The Binary Stress energy is defined as:

$$bStress(\mathrm{X}) = \sum_{\{i,j\} \in E} \|x_i - x_j\|^2 + \alpha \sum_{\{i,j\} \in V^{(2)}} (\|x_i - x_j\| - 1)^2.$$

The first term strives to put the connected vertices close to each other while the second term assigns some neutral distance to all pairs of vertices. Similar to the Stress function, the Binary Stress function can be minimized by majorization. In this case, the iterative solving equations are:

$$(M + \alpha L)\mathbf{X}(t+1) = \mathbf{b}^{X(t)}, \quad (M + \alpha L)\mathbf{Y}(t+1) = \mathbf{b}^{Y(t)},$$

where $\mathbf{b}^X, \mathbf{b}^Y \in \mathbb{R}^n$ and $M \in \mathbb{R}^{n \times n}$ are defined as:

$$\mathbf{b}^X_i = \sum_{i \neq j} \frac{x_i - x_j}{\|(x_i, y_i) - (x_j, y_j)\|}, \quad \mathbf{b}^Y_i = \sum_{i \neq j} \frac{y_i - y_j}{\|(x_i, y_i) - (x_j, y_j)\|}.$$

$$M_{ij} = \begin{cases} -1 & i \neq j \\ n-1 & i = j \end{cases} .$$

The authors also use the Barnes and Hut algorithm to increase the speed of the computation of $\mathbf{b}_i^X$ and $\mathbf{b}_i^Y$.

LinLog Model

Recently, Noack [101] has investigated the influence of the shape of attraction and repulsion energies[2] on the clustering properties of a model. It is shown that in order for a model to have a chance for revealing the clusters the repulsion energy function $f(z)$ and the attractive energy function $g(z)$ must have $\frac{f'(z)}{g'(z)} = z$. The *LinLog* model is consequently proposed using *Lin*ear attraction and *Log*arithmic repulsion energies in two versions called the node repulsion LinLog and the edge repulsion LinLog. The node repulsion LinLog is defined as:

$$\mathrm{LinLog}(\mathbf{X}) = \sum_{\{i,j\} \in E} \|x_i - x_j\| - \sum_{\{i,j\} \in V^{(2)}} \ln \|x_i - x_j\|.$$

The drawback of the node repulsion LinLog is that vertices are likely to cluster around high degree nodes. In the edge repulsion LinLog, the repulsive force of each vertex is reinforced by its degree. In other words, not only do edges cause attraction between the end vertices, but also increase the repulsion between them. As a consequence, vertices have a more uniform distribution and do not get too close to high degree nodes. The edge repulsion LinLog is defined as:

$$\mathrm{EdgeLinLog}(\mathbf{X}) = \sum_{\{i,j\} \in E} \|x_i - x_j\| - \sum_{\{i,j\} \in V^{(2)}} \deg(i)\deg(j) \ln \|x_i - x_j\|.$$

For graphs with an almost regular structure there is not much difference between the layouts of the two models. Nevertheless, when the degree distribution is uneven, the edge repulsion LinLog draws more readable layouts. A remarkable point of LinLog is that the author adopts a mathematical approach to derive some properties of the minimum energy layouts. Nevertheless, he proposes no multilevel algorithm for minimization of the energy function. As a consequence, he fails to compute and report layouts of large graphs. The previously considered multilevel algorithms do not perform well on the LinLog energy function, as they usually assume uniform edge length distribution over the layout which can be seriously violated in LinLog layouts.

3.3.2 High Dimensional Embedding

Harel and Koren [52] suggest High Dimensional Embedding (HDE) for graph drawing. The motivation behind this method is that many criteria of aesthetic drawing cannot be met adequately in a low-dimensional space. Therefore, it seems reasonable to compute a nearly optimal drawing of the graph in a high-dimensional space, and to project it into the low dimensional visual space. The projection is done using a well-known multivariate technique called Principal Component Analysis (PCA), representing the maximum possible variance of the data. In order to draw a graph

[2] Energy is a state representable by a scalar. The terms *repulsion* and *attraction* can only be used for forces which are vectorial quantities. However, with a slight abuse of notation we use them for the corresponding terms in the energy function too.

$G = (V, E)$ in the high dimensional space with m dimensions, a set of m *pivot* nodes $p_1, \cdots, p_m$ are chosen such that they are uniformly distributed over the graph. The authors mention that 50 is an adequate value for m. Each axis i of the drawing in the high-dimensional space is associated with a pivot p_i. It represents the graph from the view point of p_i. Each vertex $v \in V$ is associated with a coordinate vector $x^1(v), \cdots, x^m(v)$, such that each entry represents the graph theoretic distance of v from p_i, that is, $d_{p_i v} = x^i(v)$. The Breadth-First-Search (BFS) is used to compute the graph theoretic distance between the vertices and the pivots. The first pivot is chosen uniformly at random. Next pivots $p_i,\ i = 2, \cdots, m$, are selected such that they maximize the minimum distance from the previous pivots. This way of choosing the pivots is a 2-approximation to the problem of k-center, where k centers are to be chosen such that the longest distance of the vertices from them is minimized. This approach is very fast and its running time is linear in the number of vertices. The overall complexity of this approach is $O(m \cdot |E| + m^2 \cdot |V|)$. Although HDE has an extremely low execution time, the resulting drawings are not always satisfactory.

3.3.3 Spectral Methods

Spectral graph drawing is another prominent approach to graph drawing [74, 19, 80]. It is referred to all methods relating the coordinates of the vertices to the (generalized) eigenvectors of a matrix associated with the structure of the graph. Spectral methods have two advantages over the force-based approaches. First, they formulate the drawing problem in the form of a linear algebraic optimization problem. Consequently, unlike the force-directed minimization problems which are usually NP-hard, and can be approximated at most, standard analytical solutions to linear optimization can be applied to spectral graph drawing. Second, their running time is significantly lower than their force-directed counterparts. This asset is central for visualization of the ever-growing datasets we approach. In fact, while the most advanced versions of the multilevel force-directed algorithms take some minutes to lay out graphs with $O(10^5)$ vertices and larger, spectral methods can handle them in seconds.

Spectral drawing was first suggested in 1970 by Hall [50]. Though, it was not adopted widely until recent years. Interestingly, it is not even mentioned in some older surveys like [7, 70]. One reason for this cold start is that despite their efficiency in terms of the running time, spectral layouts are usually not as aesthetically pleasing as force-directed layouts. As a consequence, they only found their place in the literature of graph drawing when researchers were faced with huge graphs of hundreds of thousands even millions of vertices. Another drawback of spectral graph drawing is that only a very limited set of drawing objectives can be written in terms of a linear optimization problem. This issue does not arise in force-directed models, as they can reflect a wide variety of criteria by adopting different shapes of force or energy functions. The spectral graph drawing is formulated in [74] as:

$$\begin{aligned} \text{minimize} \quad & \frac{\sum_{\{i,j\}\in E} \omega_{ij} d_{ij}^2}{\sum_{i<j} d_{ij}^2}, \\ \text{subject to :} \quad & \mathrm{Var}(x^1) = \mathrm{Var}(x^2) = \ldots = \mathrm{Var}(x^p) \\ & \mathrm{Cov}(x^k, x^l) = 0, \quad 1 \le k \ne l \le p. \end{aligned} \tag{3.4}$$

The condition of equal variance is to scatter the vertices equally over all axes. It keeps different dimensions of the layout proportional with each other. The second condition ensures that the axes

are uncorrelated, so each provides new information. Minimizing Equation (3.4) is related to the common objectives of graph drawing. Namely, the vertices are prevented from overlapping by being scattered over the drawing area (maximizing the denominator). The graph structure is revealed by shortening the edges (minimizing the numerator). Since edge weights appear in the numerator, vertices connected by higher weight lie closer to each other in the resulting layout. It is proved that the above minimization problem is equivalent to:

$$\begin{aligned} &\min \frac{\sum_{k=1}^{p} (x^k)^T L x^k}{\sum_{k=1}^{p} (x^k)^T x^k}, \\ \text{subject to}: \quad &(x^k)^T x^l = \delta_{kl}, \quad k,l = 1,\ldots,p \\ &(x^k)^T \cdot \mathbf{1}_n = 0, \quad k = 1,\ldots,p. \end{aligned} \tag{3.5}$$

The second constraint is to omit the degenerate solution of $\mathbf{1}_n$. The details of the proof may be found in [74]. This is a standard linear optimization problem whose solution is the first eigenvector of L corresponding to the smallest non-zero eigenvalue:

$$Lx = \mu x. \tag{3.6}$$

This result is a corrolary of Theorem 3.2.1. Similarly, the next eigenvectors can be chosen as the next axes of the drawing.

A disadvantage of graph drawing using the eigenvectors of Laplacian is that vertices of convex subgraphs are drawn very close to each other. In addition, vertices with very low degree, notably those with only one incident edge, lie isolated from the rest of the graph. As a result, the strongly connected subgraph is placed in a small area in the center of the drawing, while most of the drawing area remains empty with isolated vertices in the frontiers. The reason of this shortcoming is there is nothing that prevents the vertices from lying close to each other. This problem can be alleviated by using the degree-normalized eigenvectors of Laplacian instead of its eigenvectors. If each vertex is weighted by its degree in Equation (3.5), we obtain:

$$\begin{aligned} &\min \frac{\sum_{k=1}^{p} (x^k)^T L x^k}{\sum_{k=1}^{p} (x^k)^T D x^k}, \\ \text{subject to}: \quad &(x^k)^T D x^l = \delta_{kl}, \quad k,l = 1,\ldots,p \\ &(x^k)^T D \mathbf{1}_n = 0, \quad k = 1,\ldots,p. \end{aligned} \tag{3.7}$$

The solution to this problem is the generalized eigenvectors of (L, D) corresponding to the smallest generalized eigenvectors:

$$Lx = \mu Dx. \tag{3.8}$$

This is a corrolary of Theorem 3.2.2. The next axes will be the next generalized eigenvectors of (L, D). It is observed that the numerator strives to put vertices with high degrees in the center of the layout to minimize their distance from the rest of the graph. On the other hand, the denominator strives to set them apart from each other. The tradeoff between these two conflicting objectives forces the vertices with high degrees to be scattered more uniformly over the layout. As a consequence, vertices with low degrees are not placed isolated from the rest of the graph.

The eigenvectors of a symmetric matrix A and the generalized eigenvectors of (A, B), where B is positive-definite, are computed through a standard method called *power iteration*. This method

initializes the first eigenvector u at random, and assigns iteratively $u \leftarrow Au$, until the change of direction of u is negligible. The next eigenvectors are computed similarly, but they are forced to be orthogonal to the previous eigenvectors. Power iteration converges to the eigenvectors of A corresponding to the *largest* eigenvalues. The computation of the generalized eigenvectors is similar, but at each iteration one sets $u \leftarrow B^{-1}Au$. Also, the orthogonality to the previous eigenvectors is changed to B-orthogonality.

Algebraic multigrid Computation of Eigenvectors

Koren et al. suggest ACE (Algebraic multigrid Computation of Eigenvectors) [80] for fast drawing of huge graphs. This algorithm applies the spectral graph drawing using the eigenvectors of the Laplacian. In order initialize sophisticatedly the eigenvectors, the authors use a multilevel algorithm to iteratively coarsen the graph. The layout of the coarser graphs are used as the initial eigenvectors of the finer ones.

Spectral Distance Embedding

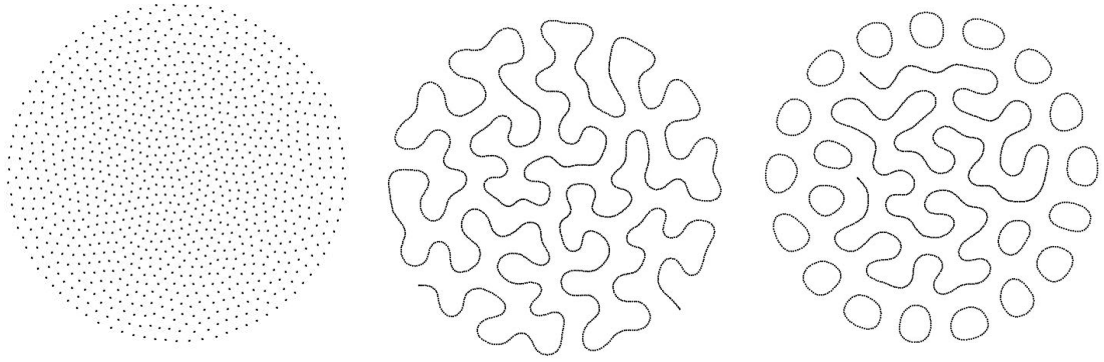
Civril et al. suggest Sampled Spectral Distance Embedding (SSDE) in [19]. SSDE relates the Euclidean distance between vertices to their graph theoretical distance through solving a set of linear equations. Namely, a constant number c of vertices of the graph are sampled, and their graph theoretical distance to all vertices is computed. These distances are held in an $n \times c$ matrix C. The rest of the distances can be approximated by a linear combination of the columns of C. Namely, if $\mathbf{L}$ is an $n \times n$ matrix where each entry $\mathbf{L}_{ij} = \mathbf{D}_{ij}^2$ ($\mathbf{D}_{ij}$ is the graph theoretical distance between i and j) it can be approximated by the multiplication of three small matrices constructed from C, that is, $\mathbf{L} \approx C\mathbf{\Phi}^+C^T$. $\mathbf{\Phi}$ is a $c \times c$ matrix corresponding to the intersection of C and C^T on $\mathbf{L}$, and $\mathbf{\Phi}^+$ is its pseudo-inverse. Once $\mathbf{L}$ is approximated, another $n \times n$ matrix $\mathbf{M}$ is built from it where $\mathbf{M} = -\frac{1}{2}\gamma\mathbf{L}\gamma$, and γ is a projection operator. The coordinates of the vertices on the layout are related to the eigenvectors of $\mathbf{M}$.

3.4 The FlexGD Energy Function

It is relatively a new finding that weighting the repulsion force of each vertex by its degree improves the quality of the drawing. In particular, drawing using the generalized eigenvectors results in more pleasing layouts than drawing using the non-generalized eigenvectors. The edge repulsion LinLog is also inspired by this finding. FlexGD goes another way to achieve the same goal in a more controllable manner. The core idea is to use *both* attractive and repulsive forces to distribute the vertices evenly over the drawing area regardless of the edges. Then, the connected vertices are forced closer to each other proportional to a parameter controlling the level of structure abstraction. In other words, one can set the vertices further by decreasing the value of the parameter (decreasing the abstraction) and vice versa.

More specifically, we replace the pairwise logarithmic repulsion energy of the LinLog model with linear-logarithmic energy, while preserving the linear attraction of edges intact. For layout $\mathbf{X}$ of a graph $G = (V, E)$ and the abstraction constant k, the FlexGD energy function is defined as:

$$\mathrm{FlexGD}(\mathbf{X}, k) \quad = \quad \sum_{\{i,j\}\in E} k\|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} (\|x_i - x_j\| - \ln\|x_i - x_j\|). \qquad (3.9)$$



(a) a disconnected graph of 1000 vertices.

(b) a line of 2000 vertices.

(c) a line of 1000 vertices with 20 circles of 50 vertices.

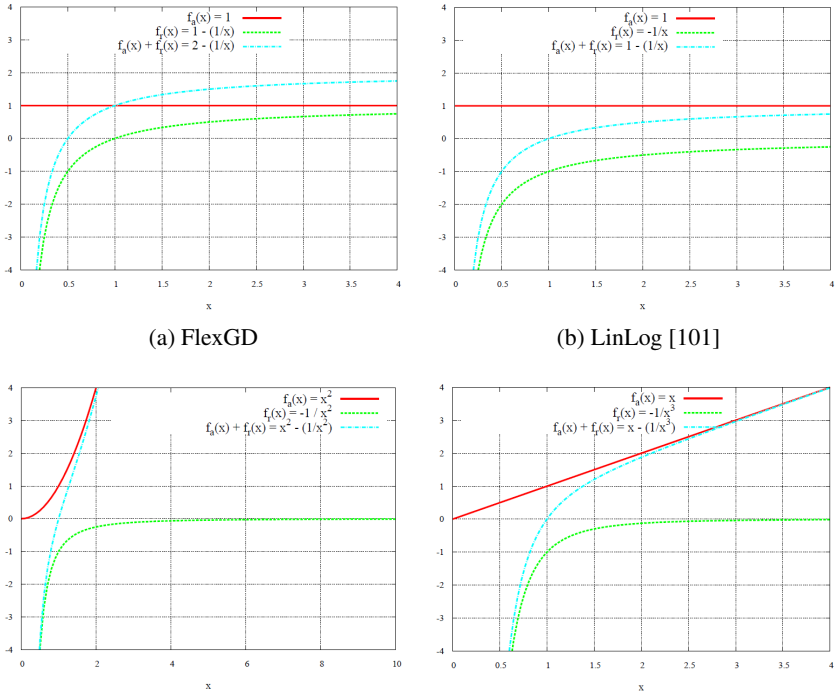
Figure 3.1: Even distribution of the graph over the drawing area.

The first term captures the graph structure by shortening the edges, while the second term distributes the vertices evenly over the drawing area.

Here, we discuss some fundamental differences of FlexGD from the previously considered models, and explain the intuition behind its behavior before presenting the more formal results. Let us first rewrite the energy function in a general form:

$$U(\mathbf{X}) = \sum_{\{i,j\} \in E} f(\|x_i - x_j\|) + \sum_{\{i,j\} \in V^{(2)}} g(\|x_i - x_j\|), \tag{3.10}$$

where $f(\|x_i - x_j\|)$ is associated with the attraction of edges, and $g(\|x_i - x_j\|)$ with the repulsion between all pairs of vertices. The minus gradients of f and g determine the attractive and repulsive forces. The form of attractive and repulsive forces determines the properties of the resulting layouts. For example, Spring Electrical results in almost even distribution of edge lengths. On the contrary, linear attraction and logarithmic repulsion energy between vertices helps the revelation of graph clusters. In models like [101, 40, 25], g is monotonously decreasing. As a result, disconnected vertices are likely to repulse each other towards infinity. It is of course possible to adjust these models for they handle disconnected graphs. For example, [39] defines a barycenter for the graph, and sets gravitational energy between the barycenter and the vertices. Another example is [38], where disconnected subgraphs are drawn separately and then packed with each other through a packing scheme. Nevertheless, such approaches complicate the implementation. In addition, using a gravitational center or a packing scheme can be questionable by itself, as such external modifications can lead to unwanted changes in the topology of the connected components which are not inherent to the graph structure. In contrast to prior models both attractive and repulsive components are present in g of FlexGD. Hence, disconnected vertices rest in a *finite neutral* distance from each other. The model can abstract the graph structure in different levels by shortening the edges less or more with respect to the neutral distance. For a simple graph of two disconnected vertices, this neutral distance occurs when $\frac{\partial g_{FlexGD}}{\partial \|x_i - x_j\|} = 0$, being $\|x_i - x_j\| = 1$. Parameter k determines how much the edges must be shorter with respect to the mean neutral distance. Figure 3.1 shows how a graph is uniformly packed within a circular drawing area regardless of its connectivity. The attractive force of edges



(a) FlexGD (b) LinLog [101]

(c) Spring Electrical [40] (d) Davidson and Harel [25]

Figure 3.2: Comparison of FlexGD forces with the forces of other models. x is the distance between the two vertices. $f_a(x)$, $f_r(x)$ and $f_a(x) + f_r(x)$ are the absolute values of the attraction force, the repulsion force and the net force that the two vertices exert on each other.

and the pairwise force exerted on a vertex i from another vertex j are obtained by derivating f and g with respect to $\|x_j - x_i\|$:

$$\vec{f_a}(i,j) = \frac{k(x_j - x_i)}{\|x_j - x_i\|} \text{ if } \{i,j\} \in E, \tag{3.11}$$

$$\vec{f_r}(i,j) = (1 - \frac{1}{\|x_j - x_i\|}) \cdot \frac{(x_j - x_i)}{\|x_j - x_i\|} \{i,j\} \in V^{(2)}. \tag{3.12}$$

The overall force exerted on i from j is then $\vec{f_a}(i,j) + \vec{f_r}(i,j)$. Figure 3.2 compares the FlexGD forces with the forces of some well-known models.

3.4.1 Properties of the Model

It is proved that adding multiplicative constants to the attractive and repulsive terms of previous force-directed models does not change the minimum energy layout, but only scales it [101, 57]. However, the minimum energy layout of FlexGD changes with k. This gives a more formal explanation of why FlexGD can draw layouts in different levels of abstraction. This claim will be investigated more in the sequel. Figure 3.3 shows the layout of an email network containing 265214 vertices and 420045 edges. Edges are not represented for more clarity. The abstraction constant is chosen as $k = 4000$ in Figure 3.3a. The clustered nature of this webmail graph is clear in this

figure. Though, one may choose to abstract the layout more at the price of viewing less details. The layout of the same graph is shown in Figure 3.3b for $k = 20000$.

Figure 3.3 also demonstrates two further assets of FlexGD layouts. Firstly, there is an empty space around clusters. It is very helpful in distinguishing the frontier between them. These clusters are particularly meaningful in social networks or web graphs, where they represent friendship groups or societies. This effect is due to the intra-distance between the vertices of a cluster being small with respect to their average distance from the rest of the graph. Consequently, they act as supernodes with high mass, exerting strong repulsion to the outside vertices pushing them further from the community. This effect increases with k, as the clusters get denser for higher k. Secondly, disconnected vertices are put towards the frontiers. This prevents them from being visually noisy to the connected components of the graph.

Here, we derive some properties of FlexGD. The goal is to understand more quantitatively how the model makes a tradeoff between the two drawing criteria, and how it separates the clusters. This analysis elaborates the influence of k on the behavior of FlexGD, and highlights its differences from the LinLog model. Theorem 3.4.1 states FlexGD finds the best compromise between maximizing the geometrical mean and minimizing the generalized arithmetic mean of all vertices. It is responsible for shortening the edges and lengthening the non-edges. If the graph contains no edges, the generalized arithmetic mean is equal to the usual arithmetic mean. The maximum of the ratio is then one, as it is a well-known fact from AM-GM inequality that the geometrical mean is always greater than or equal to the arithmetic mean. The maximum is achieved when all distances are equal. Though, in the 2-dimensional space equality is impossible for more than 3 vertices, because of geometrical constraints. Consequently, the model distributes the vertices uniformly in order to maximize the ratio by closing the two means as much as possible. This property explains why the vertices have a perfect even distribution over the drawing area in Figure 3.1a. The average distance between vertices in this example is $0.9953 \approx 1$. Figure 3.4 shows the distance distribution of vertices for the disconnected graph of Figure 3.1a. When edges reside in the graph, connected vertices are put closer to each other. The reason is they are weighted more in the generalized arithmetic mean. Therefore, their further shortening, up to some extent controlled by k, decreases the generalized arithmetic mean more than it increases the geometrical mean.

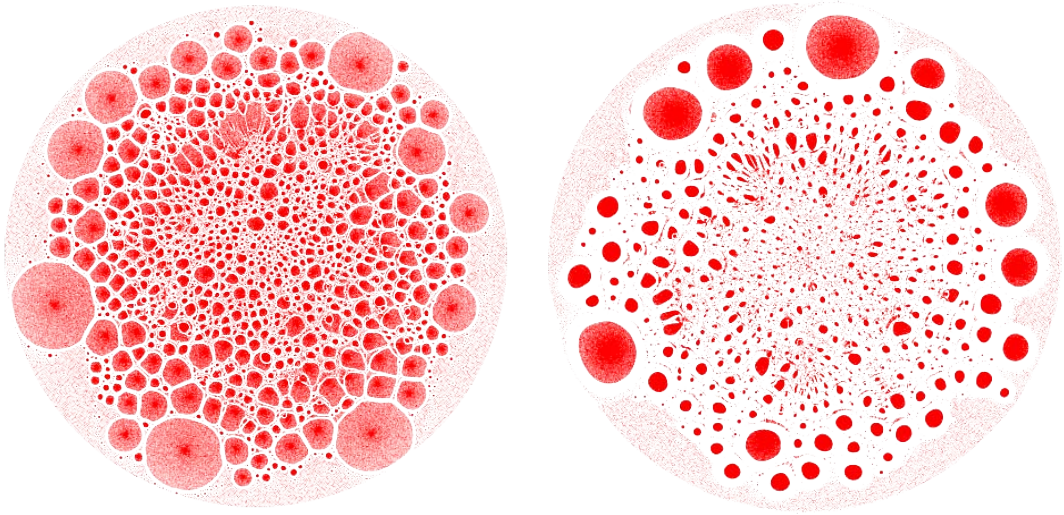
This theorem also explains why disconnected vertices are likely to go towards the frontiers. Namely, the edges are weighted k times as much in $genarith(V^{(2)}, \mathbf{X}, k)$; Since this term is in the numerator, the higher k is, the more high degree vertices are pushed towards the center of the drawing to make them close to the others, i.e. to minimize the numerator. Consequently, disconnected vertices with 0 degree go towards the frontiers.

Theorem 3.4.1 *The minimization of the FlexGD energy function is equivalent to the minimization of* $\frac{genarith(V^{(2)},\mathbf{X},k)}{geo(V^{(2)},\mathbf{X})}$.

Proof Let $\mathbf{X}^0$ be a layout with minimum FlexGD energy.
If $\sum_{\{i,j\}\in E} \|x_i^0 - x_j^0\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i^0 - x_j^0\| = c$, then $\mathbf{X}^0$ is a solution to:

$$\text{minimize}(-\sum_{\{i,j\}\in V^{(2)}} \ln \|x_i - x_j\|)$$

$$\text{subject to} \quad \sum_{\{i,j\}\in E} \|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| = c.$$



(a) email graph, $k = 4000$.

(b) email graph, $k = 20000$.

Figure 3.3: Email network from a large European research institution.

The above expression may be reformulated in the form of $\text{minimize} - \ln(geo^{|V^{(2)}|}(V^{(2)}, \mathbf{X}))$. Since $\sqrt[|V^{(2)}|]{\exp(z)}$ is an increasing function of z, the minimization of this expression is equivalent to $\text{minimize}\exp(\ln\frac{1}{geo(V^{(2)},\mathbf{X})})$. Multiplying the numerator by the constant $genarith(V^{(2)}, \mathbf{X}, k)$, and rewriting the restriction, we obtain:

$$\text{minimize}\frac{genarith(V^{(2)}, \mathbf{X}, k)}{geo(V^{(2)}, \mathbf{X})} \quad \text{subject to } genarith(V^{(2)}, \mathbf{X}, k) = \frac{c}{|E| + \left|V^{(2)}\right|}.$$

Suppose there exists a layout $\mathbf{Y}^0$ of G with minimum FlexGD energy for which $\frac{genarith(V^{(2)},\mathbf{Y}^0,k)}{geo(V^{(2)},\mathbf{Y}^0)} < \frac{genarith(V^{(2)},\mathbf{X}^0,k)}{geo(V^{(2)},\mathbf{X}^0)}$. We can always define a scaling $\mathbf{Y}^1 = \frac{c}{(|E|+\left|V^{(2)}\right|)genarith(V^{(2)},\mathbf{Y}^0,k)}\mathbf{Y}^0$ for which $genarith(V^{(2)}, \mathbf{Y}^1, k) = \frac{c}{|E|+\left|V^{(2)}\right|}$, but $\frac{genarith(V^{(2)},q^1,k)}{geo(V^{(2)},\mathbf{Y}^1)} = \frac{genarith(V^{(2)},\mathbf{Y}^0,k)}{geo(V^{(2)},\mathbf{Y}^0)} < \frac{genarith(V^{(2)},\mathbf{X}^0,k)}{geo(V^{(2)},\mathbf{X}^0)}$. We have used the fact that for every layout $\mathbf{X}$ and scaling factor λ: $genarith(V^{(2)}, \lambda\mathbf{X}^0, k) = \lambda genarith(V^{(2)}, \mathbf{X}^0, k)$. This is a contradiction as $\mathbf{X}^0$ is the minimizer of the above ratio. Hence $\mathbf{Y}^0$ does not exist and the restriction may always be removed. ∎

Theorem 3.4.2 posits that in 1-dimensional FlexGD layouts of bipartitions, the distance between the two partitions of a graph decreases with k times their density. This theorem does not generalize to more than one dimension, but remains *approximately* true for 1+ dimensional layouts of *clusterizable bipartitions*. This approximation is discussed in detail in Section 3.4.2. For graphs containing a higher number of clusters, there is in general no 2D or 3D drawing where the distance between *every* two clusters obey the same equation, without violating the triangle inequality w.r.t. a third cluster. Despite, Theorem 3.4.2 illustrates the logic behind the separation of clusters in FlexGD layouts.

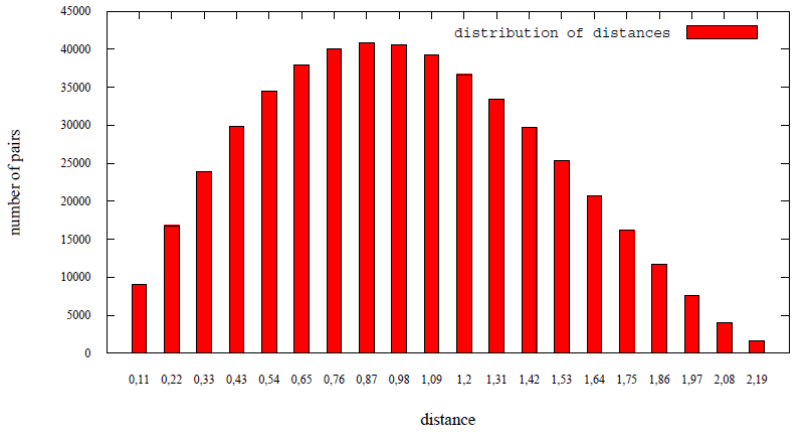


Figure 3.4: The distribution of distance between vertices in the FlexGD layout of a graph containing 1000 disconnected vertices.

Theorem 3.4.2 *Let $\mathbf{X}^0$ be a one-dimensional drawing of the graph $G = (V, E)$ with minimum FlexGD energy. Let (V_1, V_2) be a bipartition of V such that the vertices in V_1 have smaller positions than the vertices in V_2 (i.e. $\forall i \in V_1,\ \forall j \in V_2 : x_i < x_j$). Then:*

$$harm(V_1 \times V_2, \mathbf{X}^0) = \frac{1}{1 + k * density(V_1, V_2)}.$$

Proof Let $\mathbf{X}^0$ be a layout with minimum FlexGD energy. If we add $d \in \mathbb{R}$ to the coordinates of the vertices of V_1 in a way that the largest coordinate of the vertices in V_1 remains less than the smallest coordinate of the vertices in V_2, the FlexGD energy becomes:

$$\begin{aligned}\text{FlexGD}(d, \mathbf{X}^0, k) \quad = \quad & \sum_{\{i,j\} \in E_{V_1^{(2)}} \cup E_{V_2^{(2)}}} k\,|x_i - x_j| + \sum_{\{i,j\} \in V_1^{(2)} \cup V_2^{(2)}} |x_i - x_j| - \ln|x_i - x_j| \\ & + \sum_{\{i,j\} \in E_{V_1 \times V_2}} k(|x_i - x_j| + d) + \sum_{\{i,j\} \in V_1 \times V_2} |x_i - x_j| + d - \ln(|x_i - x_j| + d).\end{aligned}$$

Note that since the layout is one dimensional, $\|.\|$ and $|.|$ have the same value. Since $\mathbf{X}^0$ is a layout with minimum energy, the above function has a minimum at $d = 0$, i.e. $\text{FlexGD}'(d = 0, \mathbf{X}^0, k) = 0$. Then:

$$k\,|E_{V_1 \times V_2}| + |V_1 \times V_2| \quad = \quad + \sum_{\{i,j\} \in V_1 \times V_2} \frac{1}{|x_i - x_j|}.$$

Replacing the right side with $\frac{|V_1||V_2|}{harm(V_1 \times V_2, \mathbf{X}^0)}$ and $|V_1 \times V_2|$ with $|V_1|\,|V_2|$, the result is obtained. ∎

While Theorem 3.4.1 explains how convex subgraphs are clustered in the FlexGD layouts, Theorem 3.4.2 is responsible for separation of clusters in function of their coupling. This reminds the definition of clustering, where vertices inside a cluster must be as similar as possible, while being

Model	Minimization equivalence	One-dimensional bipartition	abstractable
LinLog [101]	$\text{minimize}\frac{arith(E,\mathbf{X})}{geo(V^{(2)},\mathbf{X})}$	$harm(V_1 \times V_2, \mathbf{X}^0) = \frac{1}{density(V_1,V_2)}$	NO
FlexGD	$\text{minimize}\frac{genarith(V^{(2)},\mathbf{X},k)}{geo(V^{(2)},\mathbf{X})}$	$harm(V_1 \times V_2, \mathbf{X}^0) = \frac{1}{1+k*density(V_1,V_2)}$	YES

Table 3.1: Summary of some properties of FlexGD and LinLog.

dissimilar from vertices of the other clusters. At this point, we would like to add that extra parameters do not give more features to the model. Theorem 3.4.3 formalizes this finding about FlexGD for a set of abstraction constants $\{k_1, k_2, k_3\}$:

Theorem 3.4.3 *The minimum of*

$$\text{FlexGD}(\mathbf{X}, k) = \sum_{\{i,j\}\in E} k_1\|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} k_2\|x_i - x_j\| - k_3 \ln \|x_i - x_j\|$$

is equal to the minimum of

$$\text{FlexGD}'(\mathbf{X}, k) = \sum_{\{i,j\}\in E} \frac{k_1}{k_2}\|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - \ln \|x_i - x_j\|,$$

up to scaling by $\frac{k_3}{k_2}$.

Proof If we scale the layout by $\frac{k_3}{k_2}$, the energy of FlexGD_{new} is:

$$\text{FlexGD}_{new}(\mathbf{X}, k) = \sum_{\{i,j\}\in E} \frac{k_1 k_3}{k_2}\|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \frac{k_3 k_2}{k_2}\|x_i - x_j\| - k_3 \ln \frac{k_3}{k_2}\|x_i - x_j\|.$$

This can be rewritten as:

$$\text{FlexGD}_{new}(\mathbf{X}, k) = k_3(\frac{k_1}{k_2} \sum_{\{i,j\}\in E} \|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - \ln \|x_i - x_j\|) - \sum_{V^{(2)}} k_3 \ln \frac{k_3}{k_2}.$$

Since k_3 is positive and $\sum_{V^{(2)}} k_3 \ln \frac{k_3}{k_2}$ is a constant, the minimum of this function is the same as the minimum of $\frac{k_1}{k_2} \sum_{\{i,j\}\in E} \|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - \ln \|x_i - x_j\|$. ∎

This theorem states that the effect of k_3 is limited to scaling, having merely a zooming effect. Furthermore, apart from its scaling effect, k_2 only changes the abstraction constant to $\frac{k_1}{k_2}$. Since every positive real value can be chosen directly as the abstraction constant, adding k_2 has no mathematical advantage. Table 3.1 compares the properties of FlexGD with the analogous results proved for LinLog in [101].

3.4.2 Distance Interpretability in 1+ Dimensional FlexGD Bipartition Layouts

In this section we explain the approximate generalizability of Theorem 3.4.2 to 1+ dimensions. The following theorem holds exactly for layouts with any number of dimensions:

Theorem 3.4.4 *Let* $\mathbf{X}$ *be a D-dimensional drawing of* $G = (V, E)$ *with minimum FlexGD energy. Let* (S_1, S_2) *be a bipartition of the drawing space by any hyperplane* H *defined by* $\sum_{l \in L} a_l y_l = b$, $L \in \binom{\{1,\cdots,D\}}{D-1}$. *If* (V_1, V_2) *is a bipartition of vertices in a way that* $\forall i \in V_1 : x_i \in S_1$, *and* $\forall j \in V_2 : x_j \in S_2$, *that is,* $\forall i \in V_1 : \sum_l a_l x_i(l) < b$ *and* $\forall j \in V_2 : \sum_l a_l x_j(l) > b$, *then:*

$$\sum_{\{i,j\} \in E_{V_1 \times V_2}} k \frac{\sum_{l=1}^{D}(x_i(l) - x_j(l))}{\|x_i - x_j\|} + \sum_{\{i,j\} \in V_1 \times V_2} \frac{\sum_{l=1}^{D}(x_i(l) - x_j(l))}{\|x_i - x_j\|} = \sum_{\{i,j\} \in V_1 \times V_2} \frac{\sum_{l=1}^{D}(x_i(l) - x_j(l))}{\|x_i - x_j\|^2}.$$

Proof If we add some distance vector $\vec{d} = (d_1, \ldots, d_D)$ to all vertices in V_1 in a way that none of them enter S_2, i.e. $\forall i \in V_1 : \sum_l a_l(x_i(l) + d_l) < b$, the energy of the new drawing is:

$$\begin{aligned}
\mathrm{FlexGD}_{new}(\mathbf{X}, k) \quad = \quad & \sum_{\{i,j\} \in E_{V_1^{(2)}} \cup E_{V_2^{(2)}}} k\|x_i - x_j\| + \\
& \sum_{\{i,j\} \in V_1^2 \cup V_2^2} (\|x_i - x_j\| - \ln \|x_i - x_j\|) + \sum_{\{i,j\} \in E_{V_1 \times V_2}} k\sqrt{\sum_{l=1}^{D}(x_i(l) - x_j(l) + d_l)^2} \\
& + \sum_{\{i,j\} \in V_1 \times V_2} (\sqrt{\sum_{l=1}^{D}(x_i(l) - x_j(l) + d_l)^2} - \ln \sqrt{\sum_{l=1}^{D}(x_i(l) - x_j(l) + d_l)^2}).
\end{aligned}$$

The partial derivative of this function with respect to d_l is:

$$\begin{aligned}
\frac{\partial \mathrm{FlexGD}_{new}(\mathbf{X}, k)}{\partial d_l} \quad = \quad & \sum_{\{i,j\} \in E_{V_1 \times V_2}} k \frac{x_i(l) - x_j(l) + d_l}{\sqrt{\sum_{l=1}^{D}(x_i(l) - x_j(l) + d_l)^2}} \\
& + \sum_{\{i,j\} \in V_1 \times V_2} (\frac{x_i(l) - x_j(l) + d_l}{\sqrt{\sum_{l=1}^{D}(x_i(l) - x_j(l) + d_l)^2}} - \frac{x_i(l) - x_j(l) + d_l}{\sum_{l=1}^{D}(x_i(l) - x_j(l) + d_l)^2}).
\end{aligned}$$

Since $\mathbf{X}$ is a layout with minimum FlexGD energy, the application of any non zero vector $\vec{d}$ must result in increase of energy. Then, the gradient vector of FlexGD_{new} must be zero when $\vec{d} = 0$, that is $\forall d_l : \frac{\partial \mathrm{FlexGD}_{new}}{\partial d_l} = 0$. Hence $\sum_{l=1}^{D} \frac{\partial \mathrm{FlexGD}_{new}}{\partial d_l} = 0$.

$$\begin{aligned}
\sum_{l=1}^{D} \frac{\partial \mathrm{FlexGD}_{new}(\mathbf{X}, k)}{\partial d_l} \quad = \quad & \sum_{\{i,j\} \in E_{V_1 \times V_2}} k \frac{\sum_{l=1}^{D}(x_i(l) - x_j(l))}{\sqrt{\sum_{l=1}^{D}(x_i(l) - x_j(l))^2}} \\
& + \sum_{\{i,j\} \in V_1 \times V_2} (\frac{\sum_{l=1}^{D}(x_i(l) - x_j(l))}{\sqrt{\sum_{l=1}^{D}(x_i(l) - x_j(l))^2}} - \frac{\sum_{l=1}^{D}(x_i(l) - x_j(l))}{(x_i(l) - x_j(l))^2}) = 0.
\end{aligned}$$

∎

For D-dimensional layouts of graphs, clusterizable to some extent, Theorem 3.4.4 leads to the following useful corollary:

Corollary 3.4.1 *Let* $\mathbf{X}$ *be a D-dimensional drawing of* $G = (V, E)$ *with minimum FlexGD energy. For any non-scaling linear transformation of the coordinate system*[3] *that partitions the vertices into* (V_1, V_2) *in a way that in the new coordinate system* $\forall i \in V_1,\ \forall j \in V_2, 1 \le l \le D : x_i(l) < x_j(l)$:[4]

$$\sum_{\{i,j\} \in E_{V_1 \times V_2}} k \frac{\|x_i - x_j\|_{\mathrm{Man}}}{\|x_i - x_j\|} + \sum_{\{i,j\} \in V_1 \times V_2} \frac{\|x_i - x_j\|_{\mathrm{Man}}}{\|x_i - x_j\|} = \sum_{\{i,j\} \in V_1 \times V_2} \frac{\|x_i - x_j\|_{\mathrm{Man}}}{\|x_i - x_j\|^2},$$

[3] This causes no change to the energy of the system.

[4] Notice such transformation does not exist for the layouts of all graphs.

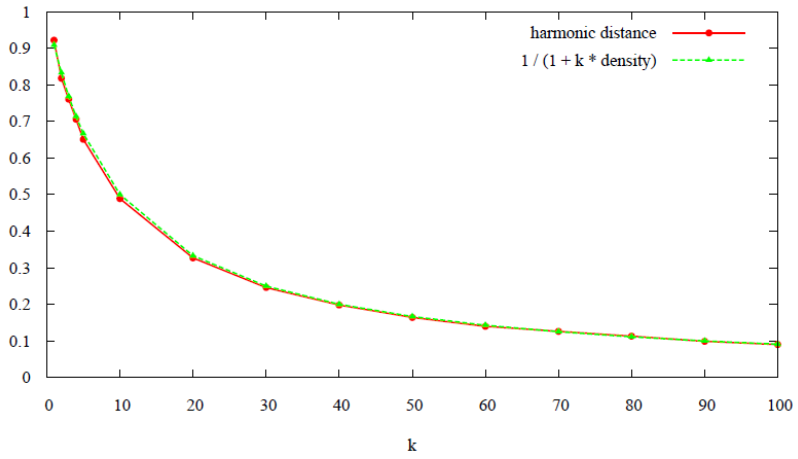


Figure 3.5: Illustration of the approximation of Theorem 3.4.2 for the 2-dimensional layout of a graph with two clusters.

where $\|x_i - x_j\|_{\text{Man}} = \sum_{l=1}^{D} |x_i(l) - x_j(l)|$ *is the* Manhattan distance *between* x_i *and* $\mathbf{x}_j$.

We know from Theorem 3.4.1 that abstraction constant may be increased to shorten the edges as much as necessary. Hence, *provided the graph is clusterizable into two convex subgraphs*, we can increase k to decrease the diameter of clusters (i.e. the maximum Euclidean distance between the pairs of a cluster) compared to their distance as much as desired. If the clusters are concentrated and far from each other, the Euclidean and Manhattan distance become almost equal. In this case we can state:

Corollary 3.4.2 *Let* $\mathbf{X}$ *be a D-dimensional drawing of* $G = (V, E)$ *with minimum FlexGD energy. If a bipartition of vertices* (V_1, V_2) *exists in a way that the diameter of* V_1 *and* V_2 *is small compared to their distance, then:*

$$harm(V_1 \times V_2, p^0) \approx \frac{1}{1 + k * density(V_1, V_2)}.$$

Proof Putting $\|x_i - x_j\| \approx \|x_i - x_j\|_{\text{Man}}$ into Corollary 3.4.1, we obtain:

$$k\,|E_{V_1 \times V_2}| + |V_1 \times V_2| \approx \sum_{\{i,j\} \in V_1 \times V_2} \frac{1}{\|x_i - x_j\|}.$$

Replacing the right side by $\frac{|V_1 \times V_2|}{harm(V_1 \times V_2, \mathbf{X}^0)}$, the result is derived. ∎

To illustrate this approximation, let us consider the simple example of a graph with two clusters of 10 vertices each. Each vertex is connected to 5 vertices of the same cluster, and there are 10 edges between the two clusters. The density between the two clusters is then equal to $10/(10 * 10) = 0.1$. Figure 3.5 shows the harmonic mean distance between the two clusters and the ratio $\frac{1}{1+k*density(V_1,V_2)}$ for different values of k. It is clear that the Theorem 3.4.2 remain correct with very small error.

3.5 Minimization of the FlexGD Energy Function

The minimum of the FlexGD energy function can be found using an iterative algorithm. In each iteration, the net force exerted on each vertex is computed. The vertices are then moved in the direction of this force by some step length until the layout change is less than some tolerance. Previous works [128, 57, 40] apply a force-directed algorithm with a cooling scheme. The algorithm starts with some global initial step length and decreases it per cycle. This scheme works well if the edge length distribution is not very uneven. In FlexGD this assumption is violated, specifically if a large level of abstraction is applied, i.e. k is set to a large value. As a result, no same value of the step length is suitable for all vertices. This issue can be treated by applying a vertex specific adaptive step length. In each iteration, the current direction of the net force exerted on a vertex is compared with its previous direction. The step length is then increased or decreased proportional to the change of direction. This scheme copes well with the non-uniform distribution of edge lengths, but has occasionally higher running time than previous simpler algorithms. Our force algorithm is given in Algorithm 3.1.

Since the force algorithm works on top of a multilevel coarsening scheme (see Section 3.5.2), it is important that the initial step length is small enough, otherwise the usefulness of the layout resulted from the previous coarsening level is lost. We compute a graph-specific initial step length with an empirical equation improvised from the following theorem:

Theorem 3.5.1 *If $\mathbf{X}^0$ is a drawing of a graph $G = (V, E)$ with minimum FlexGD energy then:*

$$k \sum_{\{i,j\}\in E} \|x_i - x_j\| + \sum_{\{u,v\}\in V^{(2)}} \|x_i - x_j\| = \left|V^{(2)}\right|.$$

Proof Suppose $\mathbf{X}^0$ is a drawing with minimum FlexGD energy. If we multiply all coordinates in $\mathbf{X}^0$ by $d \in \mathbb{R}$, the energy of the system is:

$$\mathrm{FlexGD}(d, \mathbf{X}^0, k) = k \sum_{\{i,j\}\in E} d\|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} d\|x_i - x_j\| - \ln d\|x_i - x_j\|.$$

Since $\mathbf{X}^0$ is a drawing with minimum energy, this equation has a minimum at $d = 1$, that is:

$$\begin{aligned}
\mathrm{FlexGD}'(d, \mathbf{X}^0, k) &= k \sum_{\{i,j\}\in E} \|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - \frac{\left|V^{(2)}\right|}{d}, \\
\mathrm{FlexGD}'(d = 1, \mathbf{X}^0, k) &= k \sum_{\{i,j\}\in E} \|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - \left|V^{(2)}\right| = 0.
\end{aligned}$$

∎

We can rewrite the left side of this theorem in the form of $k\,|E|\,\bar{e} + |V|^2\,\bar{d} = (k\,|E| + |V|^2)l$, where $\bar{e}$ is the mean edge length and $\bar{d}$ the mean distance between every two vertices. Hence, $l = \frac{|V|^2}{(k|E|+|V|^2)}$ is a value between $\bar{e}$ and $\bar{d}$. Dividing further by k gives some value of the order of the mean edge length. Setting the initial step length to l/k always led to satisfactory results for the graphs we tried. Further modification of the step length is done on a per vertex base through the adaptive step length

Algorithm 3.1 ForceAlgo($\mathbf{Y}$,G,k,ϵ)

$\mathbf{X} \leftarrow \mathbf{Y}$ ▷ $\mathbf{Y}$ is the initial guess from the previous round of the multilevel algorithm.
$ratio \leftarrow 2.0$ ▷ $(> 1 + \epsilon)$
$\gamma \leftarrow 0.5$
$d_0 \leftarrow 0.00000001$ ▷ small float.
$s_0 \leftarrow \frac{|V|^2}{k(k|E|+|V|^2)}$ ▷ graph-dependent initial step length.
$\forall i \in V : s_i \leftarrow s_0$
$\vec{f_u} \leftarrow random$
while $(\text{ratio} > 1 + \epsilon)$ **do**
    $BH(\mathbf{X})$ ▷ computing the Barnes and Hut tree on the current layout.
    $\text{d} \leftarrow 0.0$
    for $i \in V$ **do**
        $x_i^0 \leftarrow x_i, \vec{f_i^0} \leftarrow \vec{f_i}$
        $\forall j \in V, j \leftrightarrow i : \vec{f_i} = \vec{f_i} + \vec{f_a}(i,j)$ ▷ compute attractive forces of edges.
        $\forall j \in V, j \neq i, \vec{f_i} = \vec{f_i} + \vec{f_r}(i,j)$ ▷ compute pairwise forces. This computation is accelerated using the Barnes and Hut scheme.
        $s_i = s_i + s_i * (\frac{\vec{f_i}}{\|\vec{f_i}\|} \cdot \frac{\vec{f_i^0}}{\|\vec{f_i^0}\|}) * \gamma$ ▷ modify the step length proportional to the change of direction of the net force.
        $x_i = x_i + s_i * (\frac{\vec{f_i}}{\|\vec{f_i}\|})$ ▷ move the active vertex.
        $d = d + \left\|x_i - x_i^0\right\|$
    end for
    $ratio \leftarrow \max(\frac{d}{d_0}, \frac{d_0}{d})$ ▷ halt when the change of layout is negligible.
    $d_0 \leftarrow d$
end while

scheme. The pretty same algorithm can be used to minimize the energy function of LinLog. The only modification to do is to use an initial step length proper to LinLog. Similar to Theorem 3.5.1, it can be proved for LinLog that $\sum_{\{i,j\}\in E} \|x_i - x_j\| = \left|V^{(2)}\right|$. Therefore, one can minimize the LinLog energy function by setting the initial step length in Algorithm 3.1 to $\frac{|V|^2}{|E|}$, and use the LinLog force equations instead of those of FlexGD.

3.5.1 The Barnes and Hut Algorithm

A direct application of Algorithm 3.1 is not effective for large graphs, because it is complex as $O(|E| + |V|^2)$. A common practice in graph drawing (like [109, 81, 125]) is to decrease the complexity to $O(|E| + |V| \log |V|)$ using the Barnes and Hut scheme [5]. The idea is to speed up the calculation of pairwise forces by regrouping the nearby vertices and computing their force as a whole, provided their center of mass is far enough from the active vertex. This is done through recursively assigning the vertices to the nodes of a *quadTree*. A quadTree is a tree where each node has at most four children. There is some mass, a center and a square area associated with each node. Vertices are inserted one by one into the tree, starting from the root node. If the current node already contains a vertex, the corresponding area is divided into four squares known as *quads*. The

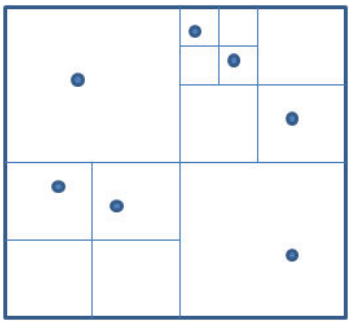


Figure 3.6: Formation of the quadTree in the Barnes and Hut algorithm.

new vertex is consequently inserted into the right quad, and the mass and the center of the parent node are updated as follows:

$$\begin{aligned} \mathbf{c} &= (m \cdot \mathbf{c} + m_i \cdot x_i)/(m + m_i), \\ m &= m + m_i, \end{aligned}$$

where m and $\mathbf{c}$ are the mass and the center of the node, and m_i and x_i the mass and the coordinates of the inserted vertex, respectively. This procedure is iterated until all vertices are inserted, and there is only zero or one vertex in each external node. Figure 3.6 illustrates the formation of a quadTree. We form the quadTree once in the beginning of each execution cycle. When computing the pairwise forces, all vertices of a node are approximated as a single vertex if $s/d < \theta$, where s is the width of the area represented by the quad of the corresponding node, and d the distance of the active vertex from the quad center. In our setting we set $\theta = 0.5$. The same scheme can be used for 3D drawings replacing the quadTree by an *ocTree* which is a tree where each node has at most 8 children.

3.5.2 A Multilevel Algorithm

The force algorithm (Algorithm 3.1) finds a *local* minimum of the energy function. Consequently, it is not very probable that it results in satisfactory drawings of large graphs, as their energy functions have many local minima. In addition, too many cycles are needed to lay a stable drawing out of the initial configuration. In fact, finding the global minimum of the energy function within reasonable complexity is the most challenging task in devising force-directed algorithms. Multilevel algorithms can greatly alleviate these problems by consecutively coarsening a graph G_0 into coarser graphs $G_1, ..., G_n$. Each coarser graph must reflect the structure of the finer one. In addition, the computation of its layout must be much less expensive than the computation of the finer graph. The layout of the coarsest graph is computed very cheaply as it is very tiny in size. The computed coordinates of the coarser graphs are then prolongated to the finer graphs. The finer graph usually needs less modifications, as it is already in a rather good shape.

Edge Collapsing (EC) [47, 129, 57] is the most widely-used coarsening strategy in graph drawing. This method works based on a Multiple Independent Edge Set (MIES). An independent edge set is a set of edges no two of which are adjacent to the same vertex. It is maximal, if adding any new edge to the set destroys the independence. MIES can be computed through a greedy algorithm. Namely, all vertices are unmatched in the beginning. An unmatched vertex is picked up at random, and is merged with one of its unmatched neighbors. As a result of this merging, the edge between them

is collapsed. Both merged vertices are then marked as matched. If the vertex has no neighbors (i.e. it is disconnected from the rest of the graph), it is marked as matched without being merged. The algorithm iterates until no vertex remains unmatched.

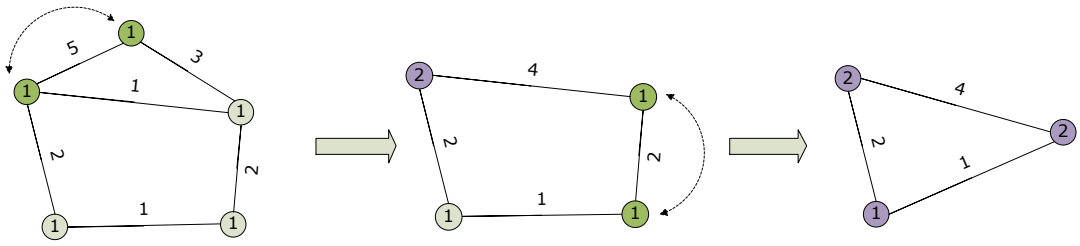
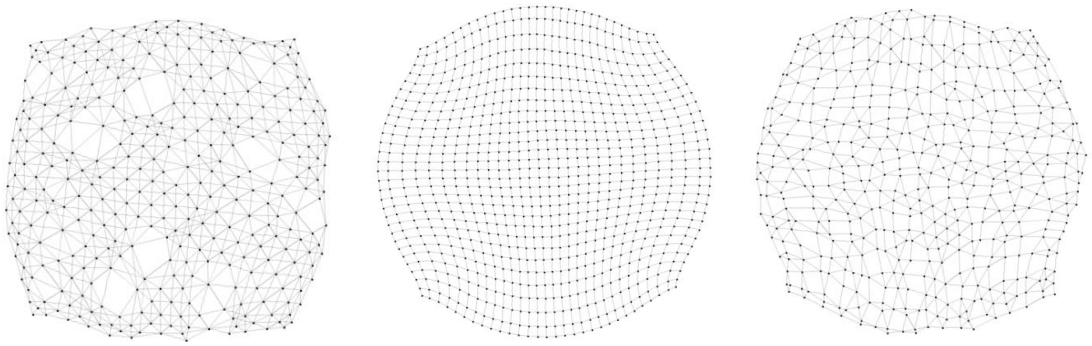


Figure 3.7: HEC coarsening a graph.

Each vertex of the coarser graph has a weight equal to the sum of the weights of the corresponding vertices in the finer graph. The edge weights of the coarser graph are initially the same as in the finer graph. If there are more than one edge between the corresponding vertices in the finer graph (as there can be up to four of them), the edge weight in the coarser graph will be the sum of the weights of the corresponding edges. Walshaw [128] matches each vertex with the neighbor having the smallest vertex weight, keeping the weight of vertices in the coarser graph as uniform as possible (Uniform Edge Collapsing). In [57], Heavy Edge Collapsing (HEC) is used, that is, the neighbor corresponding to the heaviest edge is merged. Figure 3.7 illustrates the procedure of heavy edge coarsening for a graph with 5 vertices. We implemented both Heavy Edge Collapsing (HEC) and Uniform Edge Collapsing (UEC). For our model they had almost the same results. The correspondence between the vertices of the coarser and the finer graph may be represented by a prolongation matrix. If we represent the coarser graph with G_c and the finer graph with G_f, the prolongation matrix P_f is an $n_f \times n_c$ matrix, where n_f is the number of vertices of the finer graph, and n_c the number of vertices of the coarser one. The prolongation of coordinates from the coarser graph to the finer one is done as $Y_f = P_f X_c$. Y_f is an $n_f \times d$ matrix containing the coordinates of the finer graph prolongated from the coarser graph, and X_c is the coordinates of the coarser graph in the d-dimensional space.

The success of a coarsening scheme depends on the graph topology. For example, we observed that for graphs with hollow topology, EC has difficulties escaping the local minima. In addition, for some graphs the coarsening is very slow, i.e. the number of vertices in the coarser graph is close to the number of vertices in the finer graph. Consequently, we followed [57] by adapting an alternative coarsening strategy based on a Multiple Independent Vertex Set (MIVS). A multiple independent vertex set is a set of vertices no two of which are directly connected by an edge. It is maximal, if adding any new vertex to the set leads to violation of the independence condition. When coarsening with MIVS, an edge is added between two vertices of the coarser graph, if the graph theoretical distance between the corresponding vertices of the finer graph is no more than 3. MIVS coarsens more aggressively than EC, that is, the number of vertices in the coarser graph is much smaller (usually less than 50%) than the number of vertices in the finer graph. The drawback of MIVS is that the number of edges in the coarser graph is sometimes very high. This issue increases the memory and time complexity of the algorithm. Figure 3.8 shows the result of coarsening a 30 by 30 grid through MIVS and EC coarsening strategies.



(a) MIVS coarsening (b) grid 30×30 (c) EC coarsening

Figure 3.8: A 30 by 30 grid coarsened by different strategies.

In [57], edges are computed using the Galrekin product of the prolongation matrix P and the adjacency matrix of the finer graph A_f (see [58] for more details). The Galerkin product is defined as $P^T A_f P$. We found that computing this product is expensive for large graphs. In our implementation, we first connect all the vertices in the MIVS, if their graph theoretical distance is 2. For distances equal to 3, we iterate on the edges, and for those whose neither of their end points is in the MIVS, check the neighbors of their end points. An edge is added between the two MIVS neighbors of the end points, if there is already no edge between them. This strategy works faster than computing the Galerkin product.

The prolongation of coordinates from the coarser graph to the finer graph is done as follows. If the coarser graph is issue of EC, each vertex of the finer graph corresponds to exactly one vertex of the coarser graph. In this case, the coordinates of the coarser graph are attributed directly to the corresponding vertices of the finer graph. If the coarser graph rises from MIVS, each vertex of the finer graph is either in the multiple independent vertex set or has at least one neighbor in that. In the former case, the coordinates are taken directly from the corresponding vertex of the coarser graph. In the latter, the coordinates are computed as the mean of the coordinates of the neighbors in the multiple independent vertex set. Since some vertices may have the same coordinates in the fine graph, we add some small random displacement to set them apart.

We implemented a hybrid coarsening algorithm. Our default strategy is HEC. In few cases where the result of HEC was not good enough, MIVS was used. We stop coarsening, if one of the followings happens. Firstly, the level of coarsening is more than a predefined threshold. In our setting we do not coarse more than 12 levels. Secondly, the coarsening ratio is too high. This ratio is defined as the number of connected vertices in the coarser graph by the number of connected vertices in the finer graph. We set this threshold to 0.9. Finally, the number of the remaining connected vertices is less than a minimum. We set this to 10. Algorithm 3.2 shows the complete FlexGD algorithm in a nutshell.

The execution time of the FlexGD algorithm is given in Table 3.2. Most graphs are taken from the University of Florida Sparse Matrix Collection [26]. FlexGD reveals the structure of a graph provided k is large enough. We observed choosing k as $o(\frac{|V^2|}{|E|})$ is a proper value, depending on the level of abstraction the user prefers. For graphs in the first part of Table 3.2, k has been chosen as $o(\frac{|V|^2}{|E|})$. If the graph is disconnected, the biggest component may be considered. For very

Algorithm 3.2 FlexGD(G_0,k,ϵ,$type$)

$0 \leq i \leq n : \{G_i, P_i\} \leftarrow Coarsen(G_0, type)$ ▷ type can be either HEC or MIVS.
$\mathbf{Y_n} \leftarrow random$ ▷ The layout of the coarsest graph is initialized at random.
$\mathbf{X}_n \leftarrow ForceAlgo(\mathbf{Y_n}, G_n, k, \epsilon)$
$\mathbf{Y}_{n-1} \leftarrow Prolongate(P_{n-1}, G_{n-1}, \mathbf{X}_n)$ ▷ P_n is the prolongation matrix.
$j \leftarrow n-1$
while $(j \geq 0)$ **do**
    $\mathbf{X}_j \leftarrow ForceAlgo(\mathbf{Y}_j, G_j, k, \epsilon)$
    if $(j \neq 0)$ **then**
        $\mathbf{Y}_{j-1} \leftarrow Prolongate(P_{j-1}, G_{j-1}, \mathbf{X}_j)$
    end if
    $j = j - 1$
end while

Algorithm 3.3 FA($\mathbf{Y}$,G,ϵ)

$\mathbf{X} \leftarrow \mathbf{Y}$ ▷ $\mathbf{Y}$ is the initial guess from the previous round of the multilevel algorithm.
$d_0 \leftarrow 0.00000001$ ▷ small float.
$s \leftarrow 0.3$ ▷ initial step length.
$\vec{f_u} \leftarrow random$
while $(\text{ratio} > 1 + \epsilon)$ **do**
    $BH(\mathbf{X})$ ▷ computing the Barnes and Hut tree on the current layout.
    $\text{d} \leftarrow 0.0$
    for $i \in V$ **do**
        $x_i^0 \leftarrow x_i$, $\vec{f_i^0} \leftarrow \vec{f_i}$
        $\forall j \in V, j \leftrightarrow i : \vec{f_i} = \vec{f_i} + \vec{f_a}(i,j)$ ▷ compute attractive forces of edges.
        $\forall j \in V, j \neq i, \vec{f_i} = \vec{f_i} + \vec{f_r}(i,j)$ ▷ compute pairwise forces. This computation is accelerated using the Barnes and Hut scheme.
        $x_i = x_i + s * (\frac{\vec{f_i}}{\|\vec{f_i}\|})$ ▷ move the active vertex.
        $d = d + \left\|x_i - x_i^0\right\|$
    end for
    $s = 0.9s$ ▷ decrease the step length.
    $ratio \leftarrow \max(\frac{d}{d_0}, \frac{d_0}{d})$ ▷ halt when the change of layout is negligible.
    $d_0 \leftarrow d$
end while

sparse graphs, where vertex degree distribution is very uneven, smaller values of k can also be used. Such graphs are generally issues of applications like social networks or web graphs. They are in the second part of Table 3.2. For graphs containing up to some tens of thousands of vertices, the execution time is a few minutes. This is a reasonable time considering the high quality of the layouts. All reported times include the running time of the algorithm, the time of I/O and the time of rendering the image.

It is also worth mentioning the running time of the algorithm increases with k. The reason is that

Table 3.2: Execution time of the FlexGD algorithm

Graph	$\|V\|$	$\|E\|$	k	CPU time in seconds
grid30by30	900	1740	30	11[a]
grid50by50	2500	4900	50	54
grid100by100	10000	19800	100	163
jagmesh1	936	3600	300	8
jagmesh8	1141	4303	300	5
plskz362	362	880	100	3
bcsstm07	420	3836	50	3
bcsstk24	3652	81736	200	54
G12	800	1600	500	23
G49	3000	6000	3000	160
utm1700b	1700	21509	200	46
utm3069	3060	42211	300	87
3D28984Tetra	29984	599170	1500	390
mesh2e1	360	1162	100	12
can838	838	5424	150	9
cegb2919	2919	162201	50	44
nasa1824	1824	20516	200	22
nasa2146	2146	37198	150	31
Alemder	6245	24413	1500	139
dwt1005	1005	4813	250	17
rdist3a	2398	61896	100	33
web-NotreDame	325729	1497134	1000	3142[b]
web-Stanford	281903	2312497	5000	9669
email-EuAll	265214	420045	4000	20455
webbase-1M	1000005	3105536	10000	22352

[a] Times are measured on a 2.4GHz Core2 Duo.
[b] Times are measured on a 2.5GHz Xeon E5420.

higher values of k put connected vertices closer to each other. Consequently, the Barnes and Hut algorithm divides the space into smaller quads, meaning the quadTree becomes bigger. In addition, the force algorithm needs more iterations, because the layout must be refined in smaller distances necessitating smaller values of tolerance.

Usually, the layouts of different algorithms are visually compared with each other to see which one is nicer. Of course, there are also a number of measures to compare the quality of layouts. For example Hachul and Jünger suggest in [48] evaluation criteria to measure the relative edge-crossing number and the relative edge-overlapping number (number of edges that overlap completely with each other). Nevertheless, these measures are not necessarily good indicatives for comparison between the layouts. As a consequence, the most common practice remains to show the layouts to the observers, and leave the judgment to them. Of course, this judgment may be subjective is many cases. Figure 3.9 compares the FlexGD layouts of a few graphs with the layouts of some other

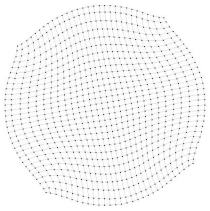
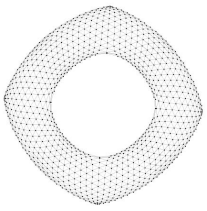
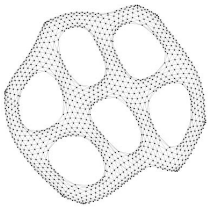
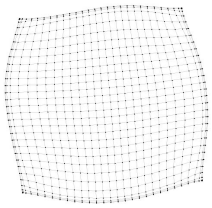
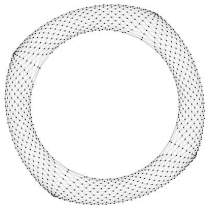
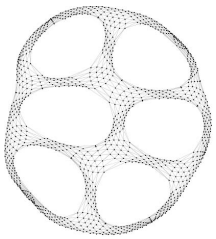
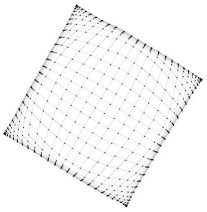
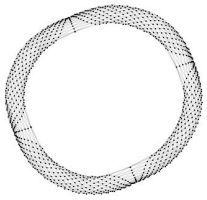
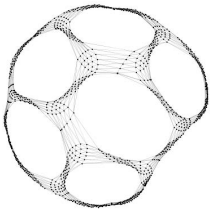
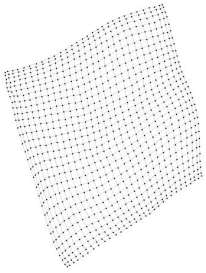
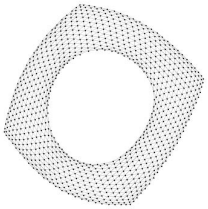
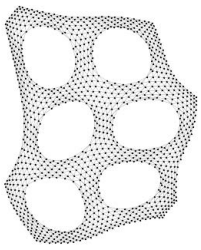
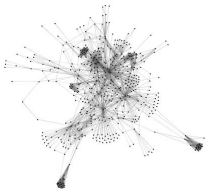
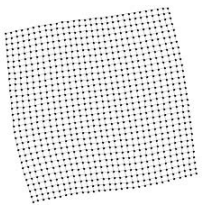
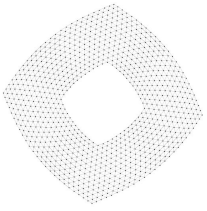
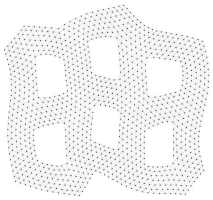
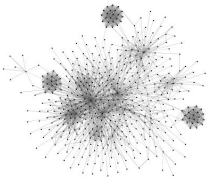
Model	grid 30by30	jagmesh1	jagmesh8	Harvard500
FlexGD (1)	 (a) $k = 30$	 (b) $k = 600$	 (c) $k = 300$	 (d) $k = 20$
FlexGD (2)	 (e) $k = 500$	 (f) $k = 1800$	 (g) $k = 900$	 (h) $k = 100$
LinLog [101]	 (i)	 (j)	 (k)	 (l)
Spring-Electrical [40]	 (m)	 (n)	 (o)	 (p)
Davidson&Harel [25]	 (q)	 (r)	 (s)	 (t)

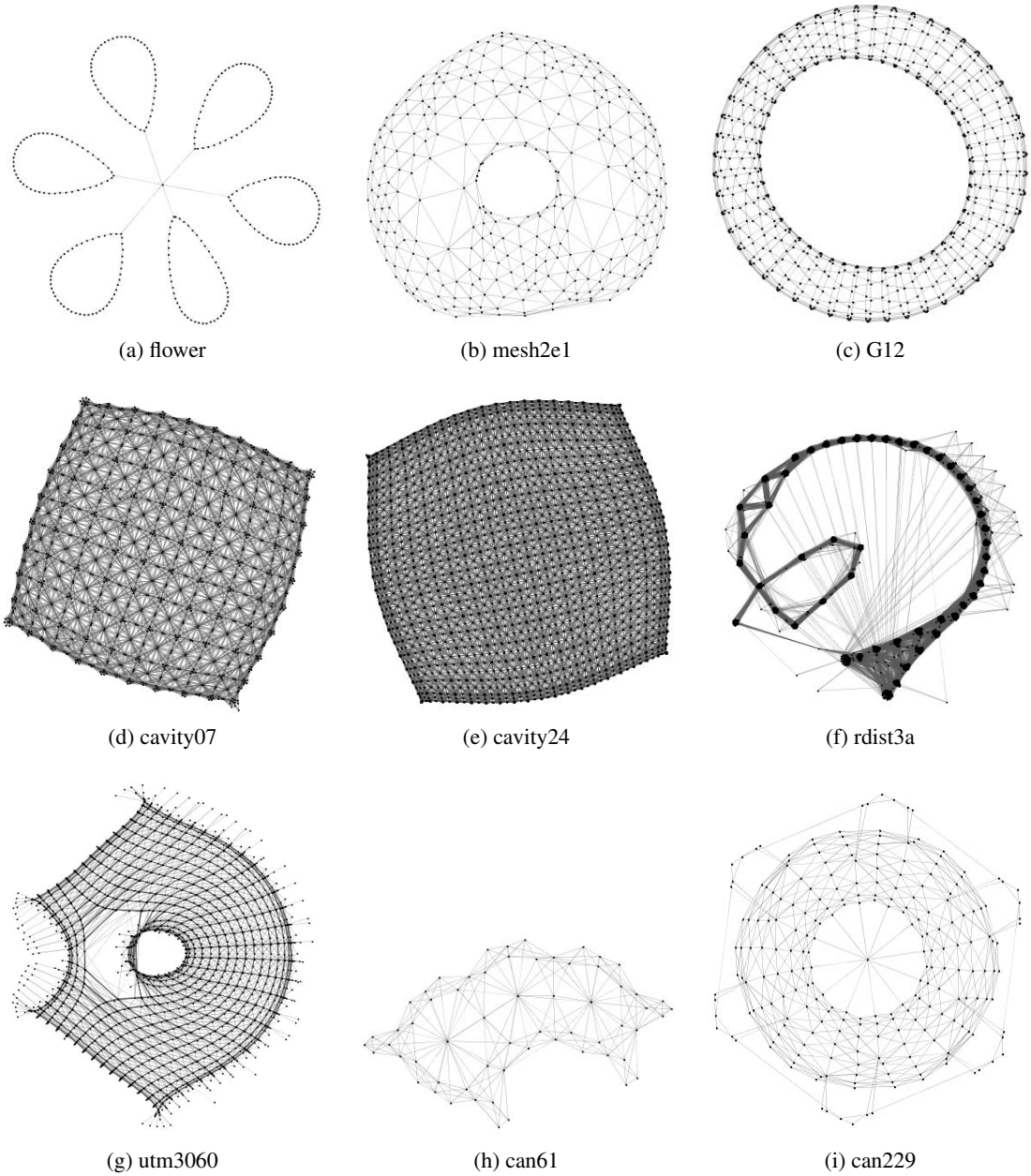
Figure 3.9: Comparison of the FlexGD layouts with other models.

force-directed algorithms. The layouts of LinLog and FlexGD are computed by Algorithm 3.1 with the corresponding initial step length and force equations. For the other models, we have used the iterative algorithm presented in Algorithm 3.3 with the same coarsening scheme described above. The symmetries are shown well in the FlexGD layouts and the frontiers are decent. The distribution of vertices in FlexGD layouts is more uniform than LinLog layouts. Notice that the frontiers of the LinLog grid layout are denser than its center. This undesirable effect is called the *peripheral effect*. For smaller values of k, the FlexGD layouts are more similar to the layouts of the conventional models, while for larger values of k, they are closer to the LinLog layouts. It is also seen that the peripheral effect increases in the FlexGD grid layout by k.

3.6 Collection of Results

Sample layouts of the FlexGD model are presented in this section. Edges and vertices of all graphs are plot with the same width and size. Further darkness of some layouts is due to the closeness of edges or vertices. It is seen that FlexGD generates layouts of very high quality. For most of the presented layouts, we are unaware of works reporting more pleasing drawings than those of FlexGD. Most graphs are taken from the University of Florida Sparse Matrix Collection [26]. It is the largest collection of real sparse matrices available. The majority of matrices that we encounter in real applications are sparse. According to [35] a matrix is sparse, if it contains so much zeros that time and memory can be saved by taking advantage of its zeros. Matrices are submitted by many research teams from various application domains. The collection is continuously growing as new recent matrices are added to the collection.

Our implementation is based on Java. We have used the Java Universal Network/Graph Framework (Jung) library [67] for graph operations and the matrix-toolkits-java (mtj) library [44] for matrix operations. This library uses BLAS and LAPACK libraries for dense and structured sparse matrix computations, and the Templates project for unstructured matrix computations.

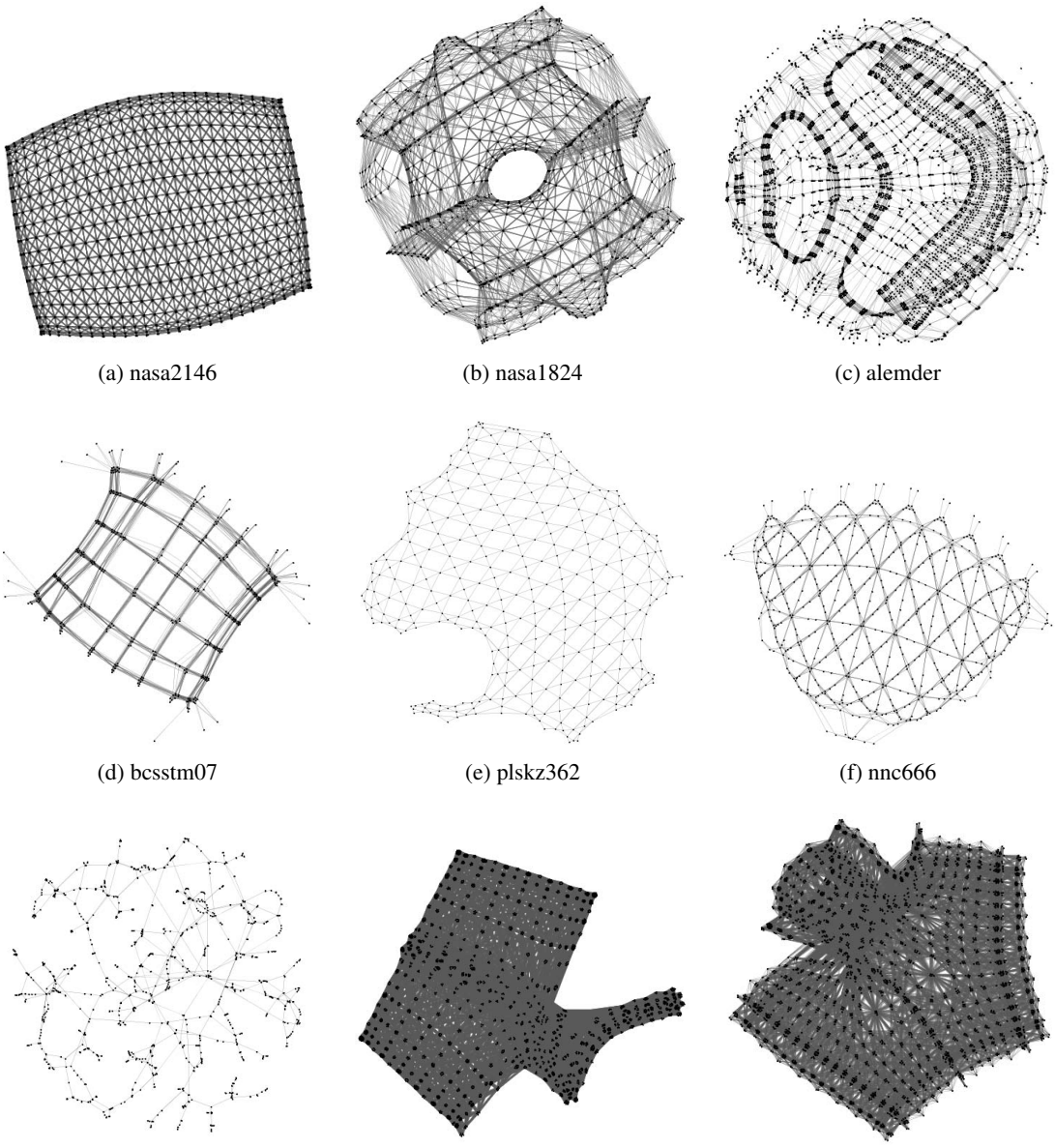


(a) flower (b) mesh2e1 (c) G12

(d) cavity07 (e) cavity24 (f) rdist3a

(g) utm3060 (h) can61 (i) can229

Figure 3.10: Sample Layouts of the FlexGD model (1).

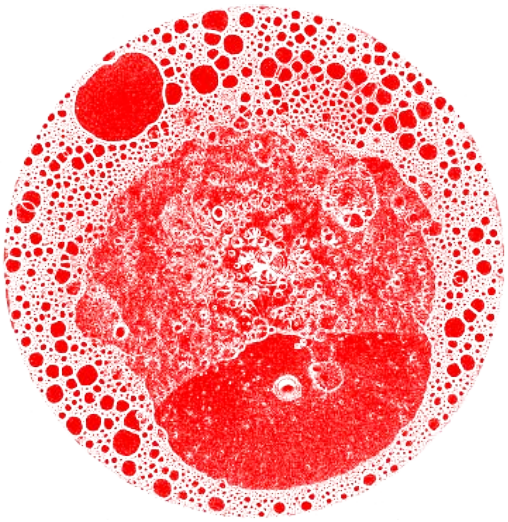


(a) nasa2146 (b) nasa1824 (c) alemder

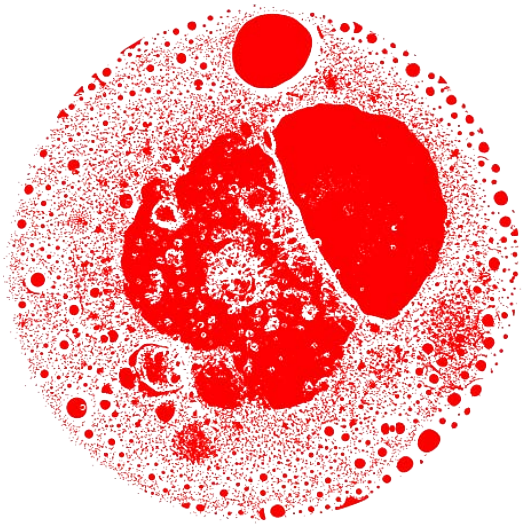
(d) bcsstm07 (e) plskz362 (f) nnc666

(g) 1138bus (h) cegb2919 (i) nasa2910

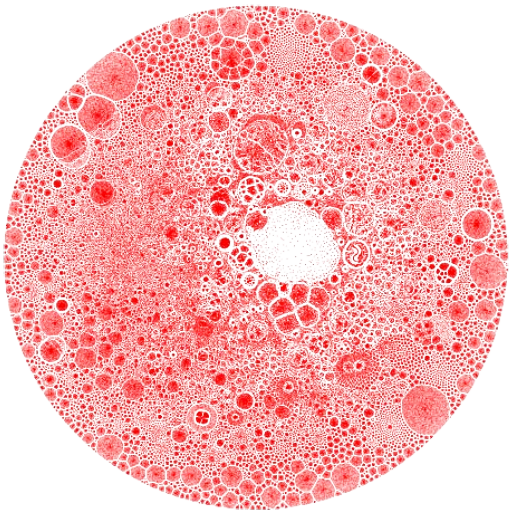
Figure 3.11: Sample Layouts of the FlexGD model (2).



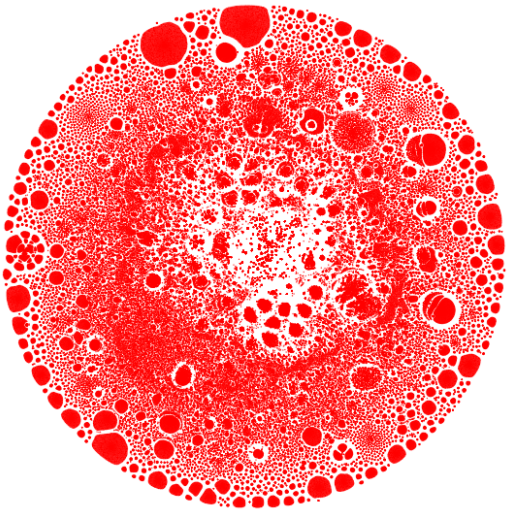
(a) Web connectivity matrix, $|V| = 1000005$, $|E| = 3105536$, $k = 10000$



(b) Web connectivity matrix, $|V| = 1000005$, $|E| = 3105536$, $k = 50000$

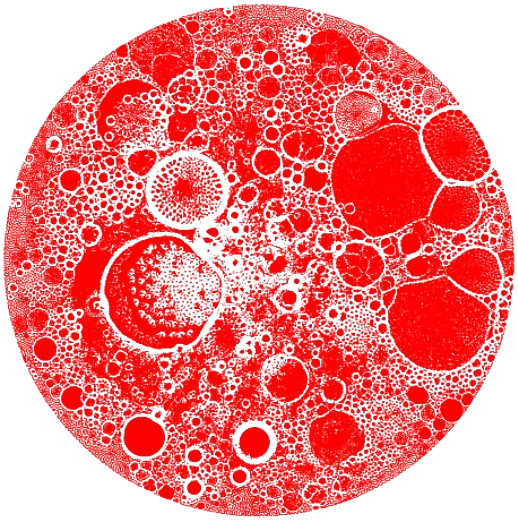


(c) Web of NotreDame University, $|V| = 325729$, $|E| = 1497134$, $k = 1000$

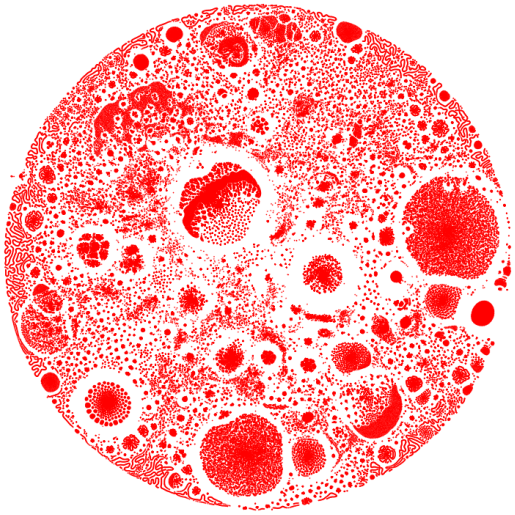


(d) Web of NotreDame University, $|V| = 325729$, $|E| = 1497134$, $k = 5000$

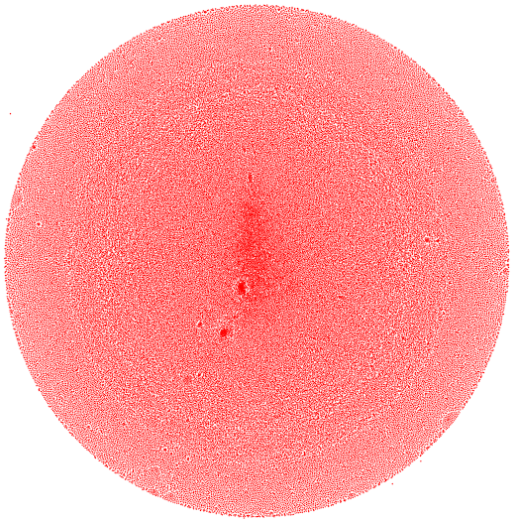
Figure 3.12: Sample Layouts of the FlexGD model (3).



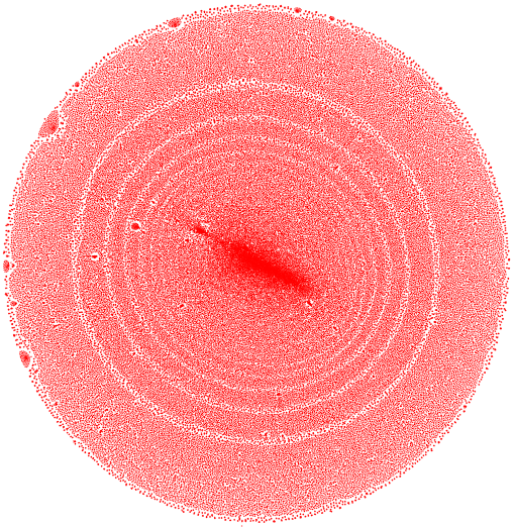
(a) Web of Stanford University, $|V| = 281903$, $|E| = 2312497$, $k = 1000$



(b) Web of Stanford University, $|V| = 281903$, $|E| = 2312497$, $k = 5000$



(c) Slashdot, $|V| = 82168$, $|E| = 948464$, $k = 200$



(d) Slashdot, $|V| = 82168$, $|E| = 948464$, $k = 1000$

Figure 3.13: Sample Layouts of the FlexGD model (4).

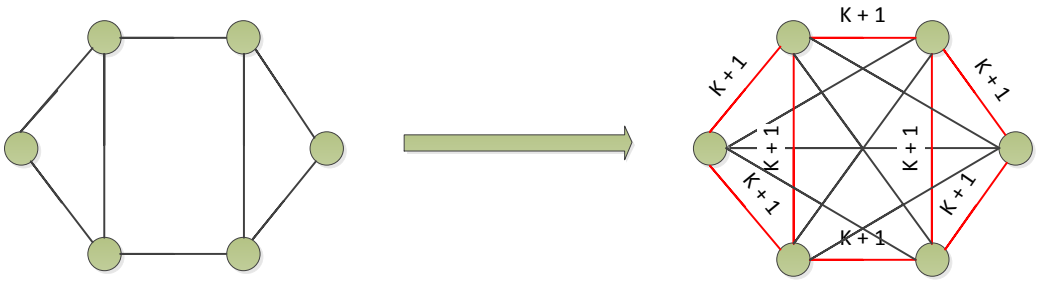


Figure 3.14: Formation of a complete weighted graph from the original graph in flexible spectral graph drawing.

3.7 Flexible Graph Drawing with the Spectral Approach

So far we have suggested FlexGD as a force-directed solution to flexible graph drawing in different levels of structure abstraction. In this section we show how a similar idea can be implemented with the spectral approach. We refer to it as FlexSpec for Flexible Spectral Approach.

In Section 3.3.3 the spectral approach was presented from an energy point of view. The spectral approach can also be characterized directly in a form which is intuitively more understandable. Here, we explain this intuition which has also been discussed in previous works on spectral graph drawing. Suppose that the coordinates of each vertex are defined as the weighted centroid of its neighbors. For vertex i, this can be written as:

$$x_i = \frac{\sum_{j \in N(i)} \omega_{ij} x_j}{deg(i)}.$$

If we write this equation in the matricial form it yields:

$$\begin{aligned} D\mathbf{X} &= A\mathbf{X}, \\ L\mathbf{X} &= 0. \end{aligned}$$

The best non-degenerate *approximative* solutions to the above equation are the first eigenvectors of L corresponding to the smallest eigenvectors. Since it is impossible to put all vertices exactly in the weighted centroid of their neighbors, one may tolerate some distance between each vertex and the centroid of its neighbors:

$$x_i - \frac{\sum_{j \in N(i)} \omega_{ij} x_j}{deg(i)} = \mu x_i. \tag{3.13}$$

This way the vertices which are further from the origin, i.e. larger values of x_i, can deviate more from the centroid of their neighbors. If this equation is written in the matricial form for all vertices it yields:

$$\begin{aligned} D\mathbf{X} - A\mathbf{X} &= \mu D\mathbf{X}, \\ L\mathbf{X} &= \mu D\mathbf{X}. \end{aligned}$$

The *exact* solutions to this equation are the generalized eigenvectors of (L, D). While in Section 3.3.3 the (generalized) eigenvectors were presented as the energy of the layout, in this type of explanation they represent the deviation from the origin.

Now, suppose that instead of putting each vertex in the weighted centroid of its neighbors, we put it in the weighted centroid of *all* vertices such that the vertex is deviated more towards its neighbors according to the parameter k. This can be written as:

$$x_i = \frac{\sum_{j \in V} \omega'_{ij}(k) x_j}{deg'(i,k)}, \quad \omega'_{ij}(k) = \begin{cases} k+1 \text{ if } \{i,j\} \in E \\ 1 \text{ otherwise} \end{cases},$$

and $deg'(i,k) = \sum_{j \neq i} \omega'_{ij}(k)$. Writing it in the matricial form, we obtain:

$$\begin{aligned} D'(k)\mathbf{X} &= A'(k)\mathbf{X}, \\ L'(k)\mathbf{X} &= 0, \end{aligned}$$

where $A'_{ij}(k) = \omega'_{ij}(k)$, $D'(k)$ is a diagonal matrix where $D'_{ii}(k) = \sum_{i \neq j} \omega'_{ij}(k)$ and $L'(k) = D'(k) - A'(k)$. Since $L'(k)$ is a Laplacian matrix itself, it can be used to represent a graph $G'(k)$ whose structure is closely related to the structure of G. $G'(k)$ is a *complete* graph with the same vertices as G. The edge weight between two vertices i' and j' of $G'(k)$ is 1, if there is no edge between the corresponding vertices i and j in G, and $(k+1)\omega_{ij}$, otherwise. Figure 3.14 shows how $G'(k)$ is constructed from G for an unweighted sample graph.

Inserting $G'(k)$ and $L'(k)$ into Equations (3.4) and (3.5), FlexSpec can be interpreted from the energy point of view. Namely, the first eigenvectors of $L'(k)$ correspond to the minimizers of the energy function:

$$\begin{aligned} & \min \frac{\sum_{h=1}^{p} (x^h)^T L'(k) x^h}{\sum_{h=1}^{p} (x^h)^T x^h}, \\ \text{subject to :} \quad & (x^h)^T x^l = \delta_{hl}, \quad h, l = 1, \ldots, p \\ & (x^h)^T \cdot \mathbf{1}_n = 0, \quad h = 1, \ldots, p, \end{aligned} \tag{3.14}$$

which is equivalent to the minimization of:

$$\begin{aligned} & \text{minimize} \quad \frac{\sum_{i<j} \omega'_{ij}(k) d_{ij}^2}{\sum_{i<j} d_{ij}^2}, \\ \text{subject to :} \quad & \text{Var}(x^1) = \text{Var}(x^2) = \ldots = \text{Var}(x^p) \\ & \text{Cov}(x^h, x^l) = 0, \quad 1 \leq h \neq l \leq p. \end{aligned}$$

With increasing k, the degree of vertices with a larger number of incident edges grows faster than those with very few incident edges. Consequently, the vertices with a smaller number of incident edges are likely to be spread more uniformly over the layout in order to minimize the numerator of Equation (3.14). Hence, in contrast to drawing using the the eigenvectors, the isolation of vertices with a small number of incident edges can be avoided in FlexSpec by increasing k. Remember the same goal is achieved in degree-normalized graph drawing by reinforcing the repulsive force of each vertex by its degree.

The generalized eigenvectors of (A, B), where A is symmetric and B positive-definite, can be found with the standard power iteration algorithm given in Algorithm 3.4. If the non-generalized

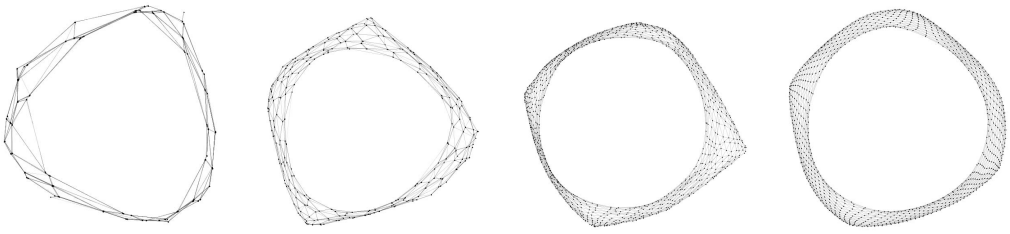
eigenvectors are needed, one can set $B = \mathbf{I}$. The power iteration converges towards the largest (generalized) eigenvectors. Since the smallest eigenvectors of C are needed, we compute the largest eigenvectors of $\tau\mathbf{I} - C$. The eigenvectors of this latter are equal to the eigenvectors of C in the reverse order. τ is the maximum Gershgorin bound. Multiplication by τ is to make sure that the computed eigenvalues are positive, and no negative energy is computed for the layout. The Gershgorin circle theorem bounds the spectrum of a matrix. It posits that every eigenvector of a matrix A lies within a disc (A_{ii}, R_{ii}), centered at A_{ii} with radius $R_{ii} = \sum_{j \neq i} |A_{ij}|$. Therefore, the largest eigenvalue of A cannot be larger than $\max_{i \in V}\{A_{ii} + \sum_{j \neq i} |A_{ij}|\}$. For the Laplacian, all eigenvalues are real-valued, and the Gershgorin discs reduce to one-dimensional boundaries.

The most time taking operation of Algorithm 3.4 is to compute $\hat{C}u_i$. It can be computed very efficiently using the sparsity of the Laplacian. Unlike L, $L'(k)$ is strictly speaking dense. However, it is practically as sparse as L, because we know all zero values of L are replaced by 1 in $L'(k)$. It can easily be shown that for any vector x, $\hat{C}'_i(k)x = k \cdot \hat{C}_i x + (n-2)x(i) + \sum_{j=1}^{n} x(j)$. Once $\sum_{j=1}^{n} x(j)$ is computed in linear time, each entry of the vector $\hat{C}'(k)x$ can be computed using the entries of $\hat{C}x$ and x. In order to have a proper initialization of the eigenvectors, we apply the HEC coarsening strategy discussed in Section 3.5.2. The computed layout of each coarser graph is used as the initial eigenvector in the next step. We observed that decreasing the value of ϵ for each finer graph improves the quality of the final layout. We then set $\epsilon \leftarrow 0.5\epsilon$ for each level of the multilevel scheme. Figure 3.15 shows the effect of k on the layouts of jagmesh1 and jagmesh8.

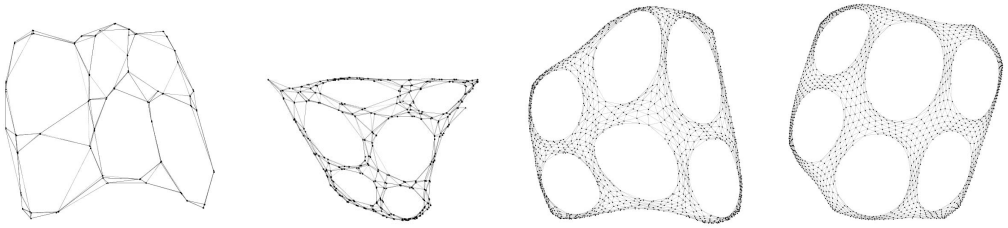
Algorithm 3.4 Power Iteration ($\{\hat{v}_i\}$,A,B,ϵ)

1: $\hat{u}_1 \leftarrow B^{\frac{1}{2}} \cdot \mathbf{1}_n$ ▷ The first known eigenvector.
2: $\hat{u}_1 \leftarrow \frac{\hat{u}_1}{\|\hat{u}_1\|}$
3: $C \leftarrow B^{-\frac{1}{2}} A B^{-\frac{1}{2}}$
4: $\tau \leftarrow Gershgorin\_Max(C)$
5: $\hat{C} \leftarrow \tau\mathbf{I} - C$ ▷ Reversing the order of eigenvectors.
6: **for** ($i = 2$ to d) **do**
7: $\quad \hat{u}_i \leftarrow B^{\frac{1}{2}} \hat{v}_i$
8: $\quad \hat{u}_i \leftarrow \frac{\hat{u}_i}{\|\hat{u}_i\|}$
9: $\quad$ condition $\leftarrow 0.0$
10: $\quad$ **while** (condition $< 1 - \epsilon$) **do**
11: $\quad\quad u_i \leftarrow \hat{u}_i$
12: $\quad\quad$ **for** ($j = 1$ to $i - 1$) **do** ▷ Orthogonalize against the previous eigenvectors.
13: $\quad\quad\quad u_i \leftarrow u_i - (u_i^T \cdot u_j)u_i$
14: $\quad\quad$ **end for**
15: $\quad\quad \hat{u}_i \leftarrow \hat{C}u_i$
16: $\quad\quad \hat{u}_i \leftarrow \frac{\hat{u}_i}{\|\hat{u}_i\|}$
17: $\quad\quad$ condition $\leftarrow \hat{u}_i \cdot u_i$ ▷ $\{\hat{u}_i\}$ are the computed eigenvectors.
18: $\quad$ **end while**
19: **end for**

A question that naturally comes to mind is how the behavior of the degree normalized versions of FlexGD and FlexSpec would be. We would like to point that these models are of no interest.



(a) $k = 0.1$ (b) $k = 1$ (c) $k = 10$ (d) $k = 100$



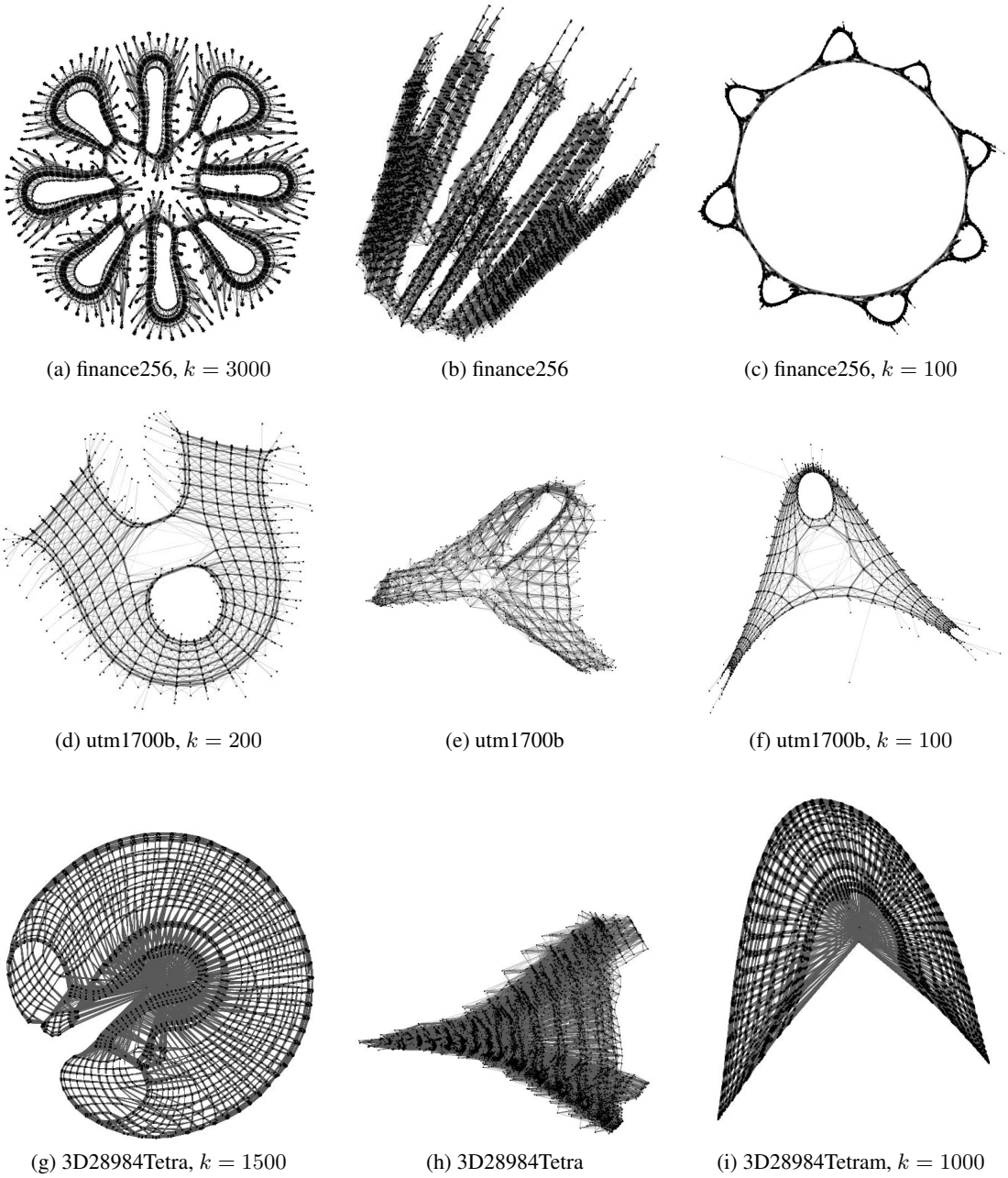
(e) $k = 0.1$ (f) $k = 1$ (g) $k = 10$ (h) $k = 100$

Figure 3.15: Influence of k on the layouts of FlexSpec.

The reason is that weighting the repulsion forces with the vertex degrees neutralizes the effect of k. More specifically in the FlexGD, the attraction force between two connected edges increases with k. On the other hand, the degree of the vertices increases with k too. Consequently, if the repulsion forces are weighted by the vertex degree (like in edge repulsion LinLog), both attraction and repulsion forces will increase with k. This prevents the parameter k from posing a reasonable control on the behavior of the model. A similar argument can be made for the degree normalized FlexSpec. For all of the graphs that we experimented, the degree normalized versions of FlexGD and FlexSpec failed to reveal the graph structure.

Figures 3.16, 3.17, 3.18, 3.19, 3.20, 3.21 and 3.22 compare the layouts of FlexGD, HDE [52] and the spectral approach with each other. The layouts of FlexSpec are computed through the power iteration and the HEC coarsening strategy. We have also observed that for large values of k (100 and higher), FlexSpec has almost the same layouts as spectral drawing with the degree-normalized eigenvectors. We refer to the latter as DN-ACE for Degree-Normalized ACE (ACE for graph drawing with non-generalized eigenvectors is discussed in [80] and was pointed in Section 3.3.3). The layouts we report are computed with FlexSpec. For graphs where the degree distribution is very uneven, both FlexSpec and DN-ACE had better results than spectral drawing with the non-generalized eigenvectors. In the case of even distribution of degrees, the performance of the three approaches is almost the same.

FlexGD HDE FlexSpec/DN-ACE



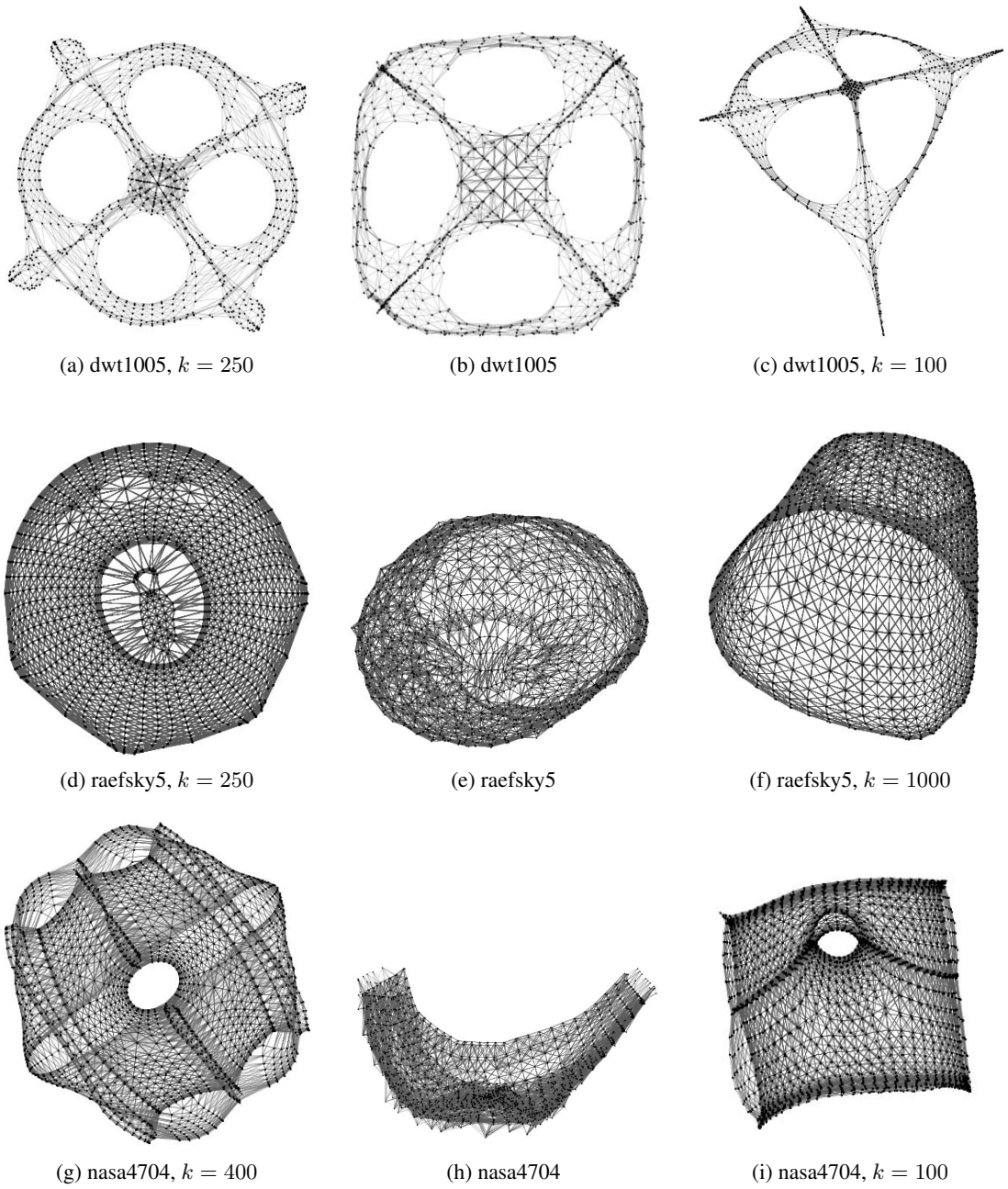
(a) finance256, $k = 3000$ (b) finance256 (c) finance256, $k = 100$

(d) utm1700b, $k = 200$ (e) utm1700b (f) utm1700b, $k = 100$

(g) 3D28984Tetra, $k = 1500$ (h) 3D28984Tetra (i) 3D28984Tetram, $k = 1000$

Figure 3.16: Comparison of drawing models (1).

FlexGD	HDE	FlexSpec/DN-ACE

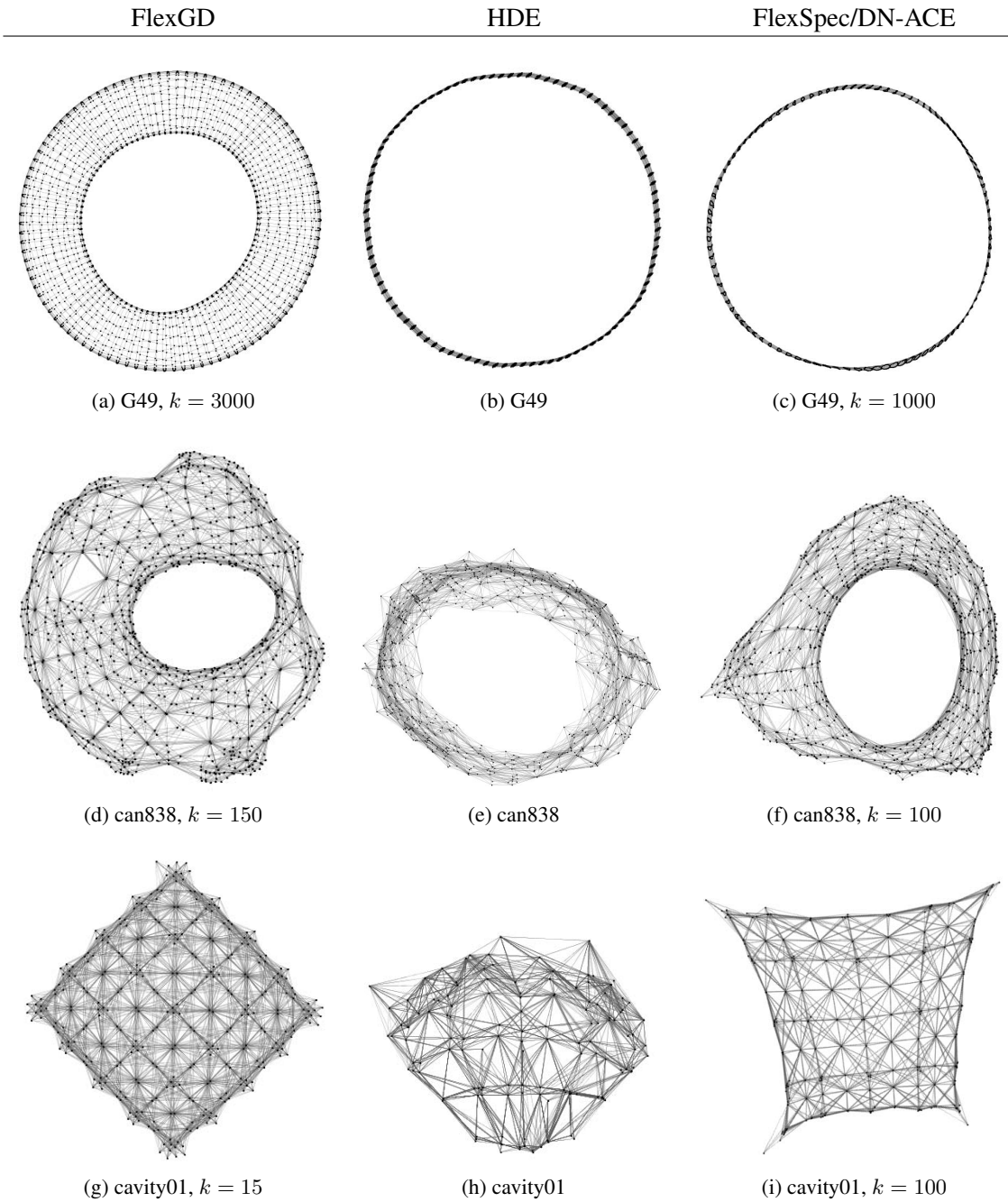


(a) dwt1005, $k = 250$ (b) dwt1005 (c) dwt1005, $k = 100$

(d) raefsky5, $k = 250$ (e) raefsky5 (f) raefsky5, $k = 1000$

(g) nasa4704, $k = 400$ (h) nasa4704 (i) nasa4704, $k = 100$

Figure 3.17: Comparison of drawing models (2).



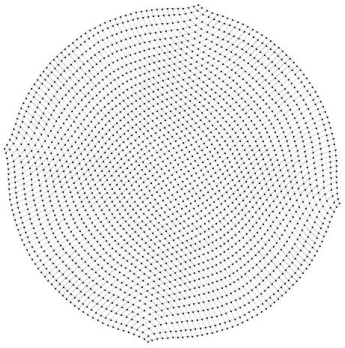
(a) G49, $k = 3000$ (b) G49 (c) G49, $k = 1000$

(d) can838, $k = 150$ (e) can838 (f) can838, $k = 100$

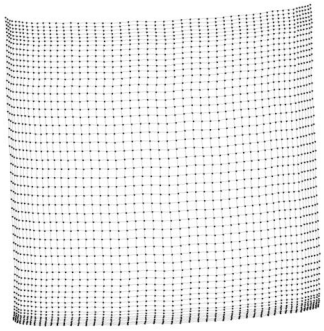
(g) cavity01, $k = 15$ (h) cavity01 (i) cavity01, $k = 100$

Figure 3.18: Comparison of drawing models (3).

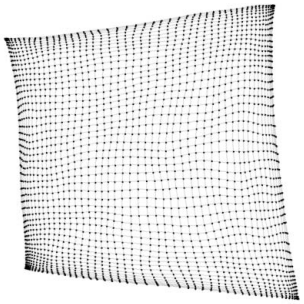
FlexGD | HDE | FlexSpec/DN-ACE



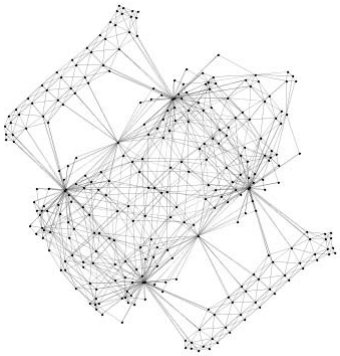
(a) grid50by50, $k = 50$



(b) grid50k50

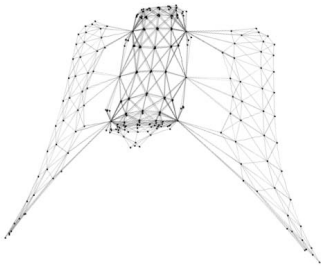


(c) grid50k50, $k = 100$

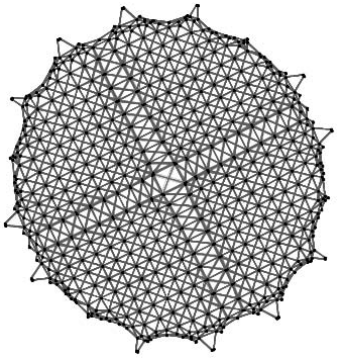


(d) can292, $k = 60$

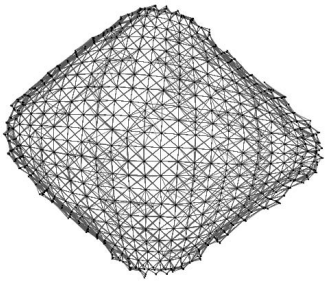
(e) can292



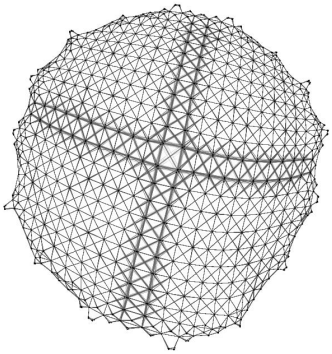
(f) can292, $k = 100$



(g) bcsstk24m $k = 200$



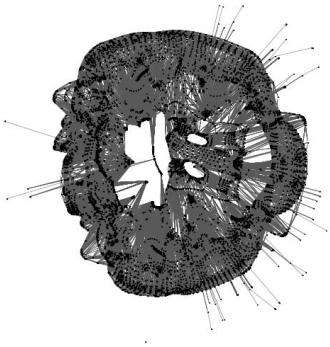
(h) bcsstk24



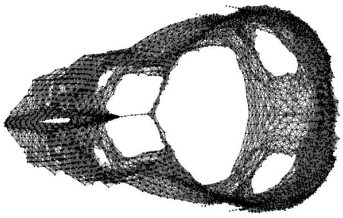
(i) bcsstk24, $k = 100$

Figure 3.19: Comparison of drawing models (4).

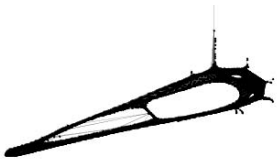
FlexGD	HDE	FlexSpec/DN-ACE



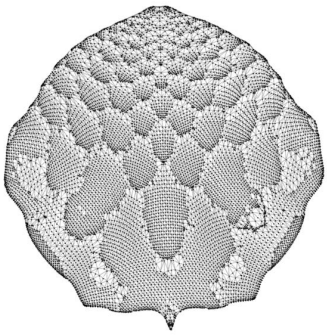
(a) bcsstk31, $k = 5000$



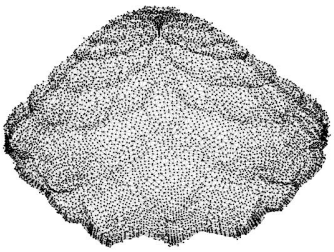
(b) bcsstk31



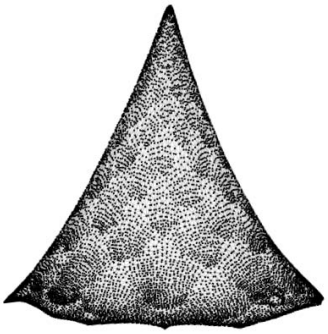
(c) bcsstk31, $k = 10000$



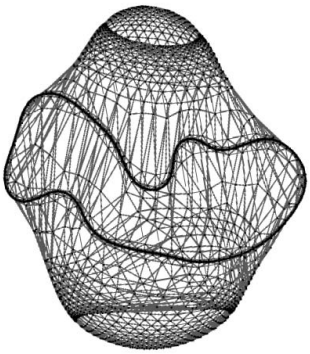
(d) crack, $k = 2000$



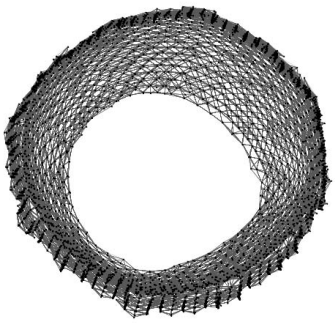
(e) crack



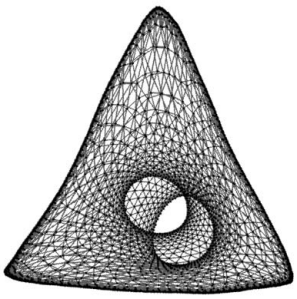
(f) crack, $k = 100$



(g) tube2, $k = 5000$



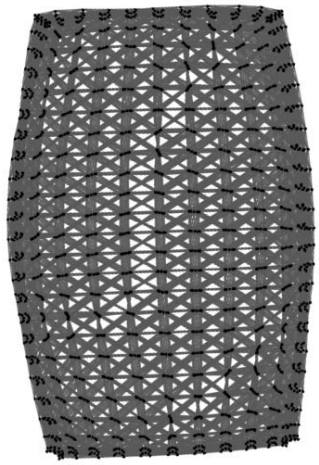
(h) tube2



(i) tube2, $k = 1000$

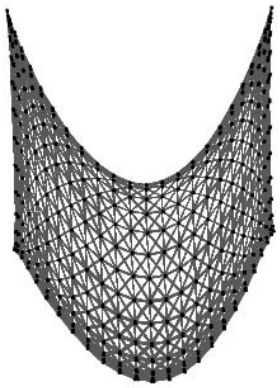
Figure 3.20: Comparison of drawing models (5).

FlexGD HDE FlexSpec/DN-ACE



(a) crystk01, $k = 200$

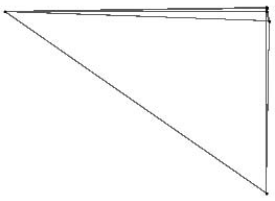
(b) crystk01



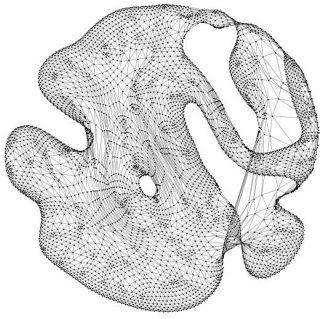
(c) crystk01, $k = 100$

(d) Harvard500, $k = 20$

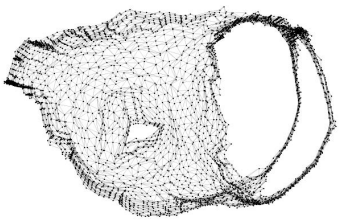
(e) Harvard500



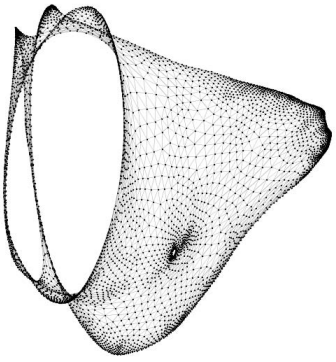
(f) Harvard500, $k = 100$



(g) 3elt, $k = 1000$



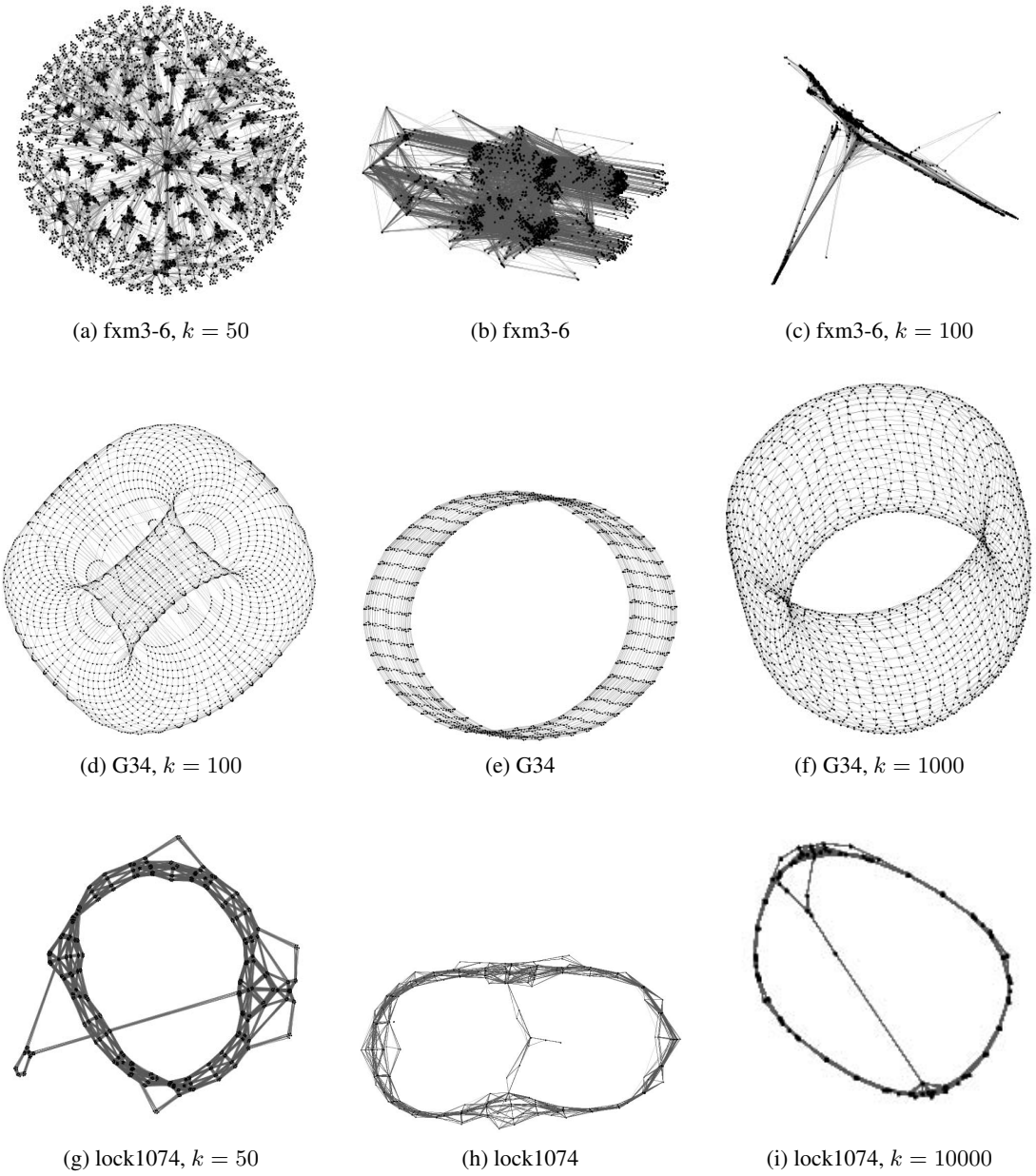
(h) 3elt



(i) 3elt, $k = 100$

Figure 3.21: Comparison of drawing models (6).

FlexGD HDE FlexSpec/DN-ACE



(a) fxm3-6, $k = 50$ (b) fxm3-6 (c) fxm3-6, $k = 100$

(d) G34, $k = 100$ (e) G34 (f) G34, $k = 1000$

(g) lock1074, $k = 50$ (h) lock1074 (i) lock1074, $k = 10000$

Figure 3.22: Comparison of drawing models (7).

3.8 Drawing Signed Graphs

In this Section we examine the problem of drawing *signed* graphs. A signed graph is a graph where edges are labeled either as positive or as negative. They appear in many applications including social networking [49] and collaborative filtering. In social networks users may express not only positive but also *negative* attitude towards each other. Likewise in collaborative filtering, ratings above a user's average score indicate positive feedback, while those below the average are representative for negative feedback. Negative interactions can be modeled by integrating the notion of edge sign into the graph. An in-depth study of signed graph visualization is of practical interest, specially because the actual graph visualization tools are oblivious to edge sign. While numerous works have studied unsigned graphs from different perspectives (ex. community detection, clustering, visualization, etc.), there are less researches about signed graphs [100]. Some examples are [15, 85] where the authors address the problem of edge sign prediction in signed social networks, and study some of their global properties. [86] investigates prediction, clustering and visualization in signed graphs through spectral analysis.

In contrast to [86], we adopt a force-directed approach to draw signed graphs. Force-directed models, including FlexGD, have in common that they interpret all edges as a measure of *attraction* between vertices. As a consequence, in their ordinary form, they cannot differentiate between positive and negative edges, and place connected vertices close to each other regardless of edge sign. Fortunately, they are flexible enough to model signed edges with few modifications. Specifically, negatively connected vertices may be excessively set apart by adding extra repulsive force between them. The addition of the repulsive term results in changes in the properties of the minimum energy layout. These changes depend on the shape of the repulsive force. We will show what modifications the linear and logarithmic repulsive energy cause to FlexGD and LinLog. We also discuss the construction of signed models through a primary-double approach and compare layouts of different signed force-directed models.

3.8.1 Force-directed Models For Drawing Signed Graphs

Let us define the general equation of force-directed models for signed graphs as:

$$U_s(\mathrm{X}) = \sum_{\{i,j\}\in E^+} f(\|x_i - x_j\|) + \sum_{\{i,j\}\in E^-} h(\|x_i - x_j\|) + \sum_{\{i,j\}\in V^{(2)}} g(\|x_i - x_j\|),$$

where $f(\|x_i - x_j\|)$ is associated with the attraction of positive edges, $h(\|x_i - x_j\|)$ with the repulsion of negative edges, and $g(\|x_i - x_j\|)$ with the pairwise energy between every pair of vertices. For connected graphs, it must be that $\exists M \in \mathbb{R}$, such that $\forall z \geq M$, $|h(z)|$ grows with equal or slower rate than $f(z) + g(z)$, otherwise the function will have no finite minimum. For FlexGD, both linear and logarithmic energy functions are correct design choices for $h(z)$. The signed version of FlexGD with logarithmic repulsion energy is defined as:

$$\mathrm{FlexGD}_s^{log}(\mathbf{X}, k_1, k_2) = \sum_{\{i,j\}\in E^+} k_1\|x_i - x_j\| - \sum_{\{i,j\}\in E^-} k_2 \ln\|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - \ln\|x_i - x_j\|.$$

Similarly, linear repulsion energy can be used:

$$\mathrm{FlexGD}_s^{lin}(\mathbf{X}, k_1, k_2) = \sum_{\{i,j\}\in E^+} k_1\|x_i - x_j\| - \sum_{\{i,j\}\in E^-} k_2\|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - \ln\|x_i - x_j\|.$$

Model	Fixed length equation
FlexGD_s^{log}	$k_1 \sum_{\{i,j\}\in E^+} \|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| = k_2 \left\|E^-\right\| + \left\|V^{(2)}\right\|.$
FlexGD_s^{lin}	$k_1 \sum_{\{i,j\}\in E^+} \|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - k_2 \sum_{\{i,j\}\in E^-} \|x_i - x_j\| = \left\|V^{(2)}\right\|.$
LinLog_s	$\sum_{\{i,j\}\in E^+} \|x_i - x_j\| = k \left\|E^-\right\| + \left\|V^{(2)}\right\|.$
	Minimization equivalence
FlexGD_s^{log}	$\mathrm{minimize} \frac{genarith^+(V^{(2)},\mathbf{X},k_1)}{gengeo^-(V^{(2)},\mathbf{X},k_2)}$
FlexGD_s^{lin}	$\mathrm{minimize} \frac{genarith^{\pm}(V^{(2)},\mathbf{X},k_1,k_2)}{geo(V^{(2)},\mathbf{X})}$
LinLog_s	$\mathrm{minimize} \frac{arith(E^+,\mathbf{X})}{gengeo^-(V^{(2)},\mathbf{X},k)}$
	One-dimensional bipartition
FlexGD_s^{log}	$harm(V_1 \times V_2, \mathbf{X}^0) = \frac{1+k_2 density^-(V_1,V_2)}{1+k_1 density^+(V_1,V_2)}$
FlexGD_s^{lin}	$harm(V_1 \times V_2, \mathbf{X}^0) = \frac{1}{1+k_1 density^+(V_1,V_2)-k_2 density^-(V_1,V_2)}$
LinLog_s	$harm(V_1 \times V_2, \mathbf{X}^0) = \frac{1+k*density^-(V_1,V_2)}{density^+(V_1,V_2)}$

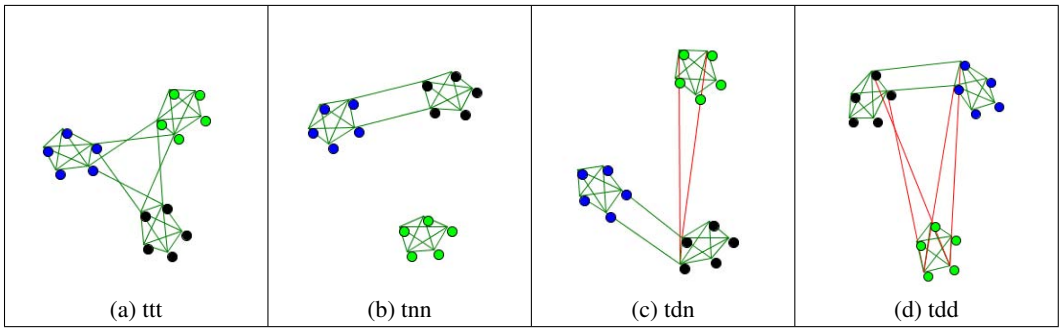
Table 3.3: Properties of the signed versions of FlexGD and LinLog.

For LinLog only logarithmic repulsion energy may be applied:

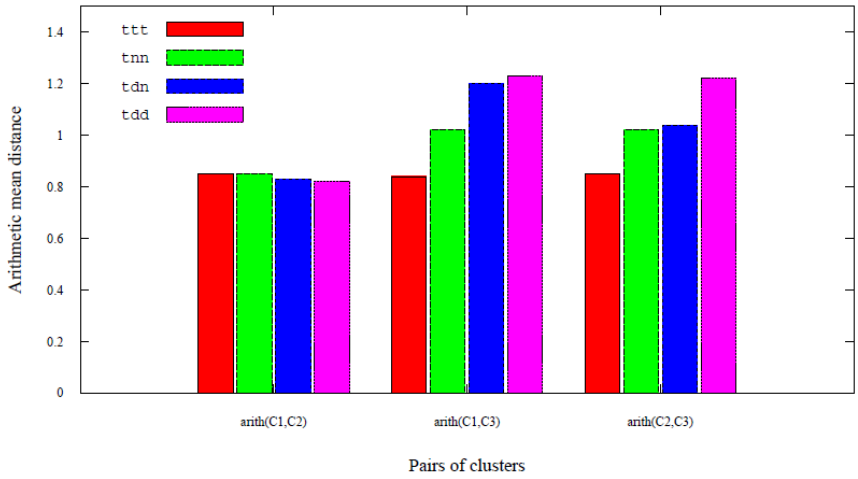
$$\mathrm{LinLog}_s(\mathbf{X}, k) = \sum_{\{i,j\}\in E^+} \|x_i - x_j\| - \sum_{\{i,j\}\in E^-} k \ln \|x_i - x_j\| - \sum_{\{i,j\}\in V^{(2)}} \ln \|x_i - x_j\|.$$

Signed versions of FlexGD can handle disconnected graphs, while in the signed version of LinLog, the graph must be connected through positive edges. It can be shown in a similar way to Theorem 3.4.3 that extra multiplicative constants do not give extra features to the models. Furthermore, the same techniques as in Section 3.4.1 can be used to derive the properties of the minimum energy layouts of the abovementioned signed models. Table 3.3 compares these properties. In the equivalent minimization problem of FlexGD_s^{lin}, $genarith^{\pm}(V^{(2)}, \mathbf{X}, k_1, k_2)$ is always positive. This is the legacy of the fixed length equation for FlexGD_s^{lin}. If the graph contains no negative edges the signed models arrive at the ordinary definition of their unsigned versions.

In both FlexGD_s^{log} and FlexGD_s^{lin}, positive edges contribute the same way to the generalized arithmetic mean which appears in the numerator. As a consequence, the model shortens the positive edges to put connected vertices close to each other. Meanwhile, vertices with a high positive indegree are put closer to the center of the layout such that they keep a reasonable distance from the other vertices. The main difference between FlexGD_s^{log} and FlexGD_s^{lin} is in the shape of contribution of the negative edges to the minimization problem. In FlexGD_s^{log}, negative edges appear in the denominator, while in FlexGD_s^{lin} they are linearly subtracted from the positive edges in $genarith^{\pm}(V^{(2)}, \mathbf{X}, k_1, k_2)$, which is in the numerator. Consequently, both models are likely to lengthen the negative edges, and put vertices with high negative indegree towards the frontiers to make them further from the rest of the graph. However, FlexGD_s^{lop} considers the *ratio* between the positive edge length and the negative edge length, while FlexGD_s^{lin} considers their *difference*. The very same effect is also seen in the one-dimensional bipartitions of the two models. Similar to Section 3.4.2, it can be proved that these equations are approximately true for bipartition layouts with more than one dimension.



(a) ttt (b) tnn (c) tdn (d) tdd



(e) Inter-cluster distance.

Figure 3.23: Visualization of trust (t), neutral (n) and distrust (d) between 3 clusters of friends using FlexGD_s^{log} with $k_1 = k_2 = 3$. Cluster 1 is black, cluster 2 blue and cluster 3 green.

We study the behavior of the presented signed models on a number of synthetic graphs and one real-world tiny graph. Unfortunately, real-world examples of signed graphs are far less available than those of unsigned graphs.

Visual Representation of the Structural Balance Theory

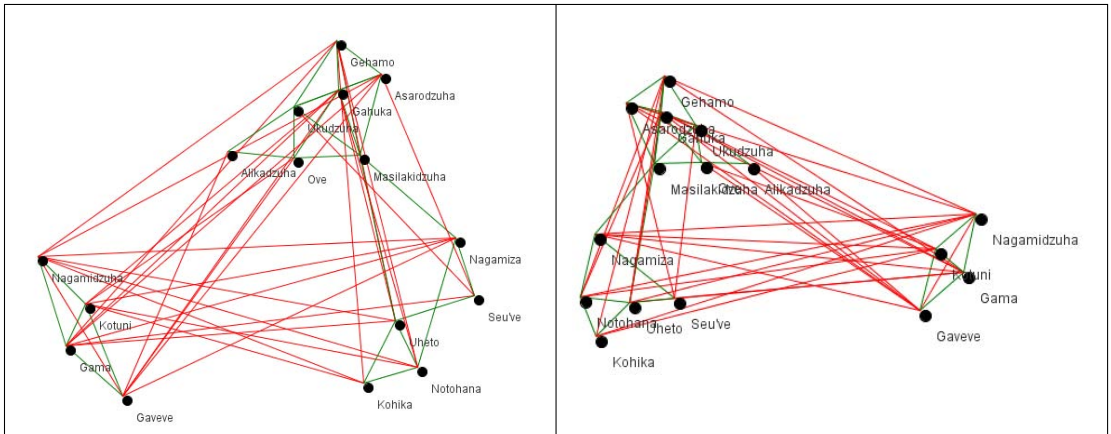
We investigate the conformance of FlexGD_s^{log} with the structural balance theory [53]. This theory originated in social psychology by F. Heider. It posits that triangles with three positive edges, or two negative edges and one positive edge are more likely to exist between three individuals. Hence, they should be more frequent in real human communities. In other words, "The friend of my enemy is my enemy" and "The friend of my friend is my friend" are more plausible than "The friend of my enemy is my friend" and "the enemy of my enemy is my enemy".

Instead of taking three individuals, we take the more general case of a graph with three communities (clusters). Figure 3.23 shows the FlexGD_s^{log} layouts of these clusters in different states of trust and distrust. Each cluster is a clique with 5 vertices. The parameters of the model are chosen as $k_1 = k_2 = 3$. Clusters 1, 2 and 3 are abbreviated with $C1$, $C2$ and $C3$, and are represented

in black, blue and green, respectively. *t* stands for *trust*, *n* for *neutral* attitude and *d* for *distrust* between two clusters. In Figure 3.23a, all clusters trust each other. The distance between all of them is 0.85 which is less than the neutral distance of 1.0. In Figure 3.23b, $C1$ and $C2$ trust each other, but have no link with $C3$. It is seen that their arithmetic distance remains as 0.85, while their distance form $C3$ increases to 1.02, which is about the neutral distance. In Figure 3.23c, $C1$ and $C2$ trust each other, but $C1$ distrusts $C3$. The distance between $C1$ and $C3$ is 1.2, being longer than the neutral distance. Furthermore, $C1$ gets slightly closer to $C2$ ($arith(C1, C2) = 0.83$), while $C2$ gets slightly further from $C3$ ($arith(C2, C3) = 1.04$) than in Figure 3.23b. This suggests that the distrust relation between two clusters makes the friends of each get slightly further from the other. In Figure 3.23d where both $C1$ and $C2$ distrust $C3$, their distance from $C3$ increases to 1.23 and 1.22, while their distance from each other decreases to 0.82. In other words, having a common enemy makes two friends even closer to each other. The inter-cluster distances in the abovementioned cases are represented in Figure 3.23e. Similar effects can be seen with FlexGD_s^{lin}.

Tribal Groups of New Guinea

A more realistic example is the tribal groups of Eastern Central Highlands of New Guinea. This graph represents 16 tribes being in war or peace state with each other. Each cluster has friend or foe relationship with others, but there is no foe state inside the same cluster. The corresponding layouts are given in Figure 3.24 for the two signed versions of FlexGD. Note that neither of the examples of this section could be drawn using the signed version of LinLog, because they are all disconnected with respect to the positive edges.



(a) FlexGD_s^{log}, $k_1 = k_2 = 3$ (b) FlexGD_s^{lin}, $k_1 = k_2 = 3$

Figure 3.24: The tribal groups of Eastern Central Highlands of New Guinea. Three higher order clusters described in [49] are revealed. Positive edges are in green, while negative edges are plot by red lines.

3.8.2 Dual-Primary Construction of Signed Energy Functions

Force-directed models can be generalized to the case of signed graphs through defining a dual model and summing it with the primary. The idea of the dual model is to interpret edges as a measure of *repulsion*. Specifically, all vertices attract each other, except if there is an edge between them. For

layout $\mathbf{X}$ of an unsigned graph $G = (V, E^-)$ with *repulsive* edges, this dual model has the general form of:

$$U_{Dual} = \sum_{\{i,j\} \in V^{(2)}} f(\|x_i - x_j\|) + \sum_{\{i,j\} \in E^-} g(\|x_i - x_j\|),$$

where $f(\|x_i - x_j\|)$ represents the attraction energy between every pair of vertices, and $g(\|x_i - x_j\|)$ the repulsion energy between connected vertices. The particular advantage of this model is to separate clusters of vertices based on their common dislikes. In a connected bipartite graph for example, the two partitions of vertices are plot apart from each other, with the vertices of each partition overlapping on the same point. Though, drawings of this model do not conform with visualization requirements, as disconnected vertices may be extremely close, even overlapping as already mentioned. Furthermore, in contexts like social networks, users with no interaction must not be interpreted as close friends. As a consequence, this model is not of interest on its own. However, its linear combination with its primary leads to a model handling both positive and negative edges, and representing acceptable properties. The behavior of the resulting model will be a compromise between the behavior of its building blocks. For a signed graph $G_s = (V, E)$ we define the dual-primary signed energy function as:

$$\begin{aligned} U_{dp}(\mathbf{X}) \quad = \quad & \sum_{\{i,j\} \in E^+} f(\|x_i - x_j\|) + \sum_{\{i,j\} \in E^-} g(\|x_i - x_j\|) + \\ & \sum_{\{i,j\} \in V^{(2)}} f(\|x_i - x_j\|) + g(\|x_i - x_j\|). \end{aligned} \tag{3.15}$$

To have an intuition into the behavior of the model, let us consider the simple example of a graph with two vertices, say i and j. If the vertices are disconnected, the minimum energy of the system is a solution to $f(\|x_i - x_j\|) + g(\|x_i - x_j\|) = 0$. In case they are positively or negatively connected, the minimum energy layout would correspond to $2f(\|x_i - x_j\|) + g(\|x_i - x_j\|) = 0$ or $f(\|x_i - x_j\|) + 2g(\|x_i - x_j\|) = 0$, respectively. If we set $f(\|x_i - x_j\|) = \|x_i - x_j\|$ and $g(\|x_i - x_j\|) = -\ln(\|x_i - x_j\|)$, the solutions to these equations would be $\|x_i - x_j\| = 1$, $\|x_i - x_j\| = 0.5$ and $\|x_i - x_j\| = 2$, for disconnected, positively connected and negatively connected vertices, respectively. It is seen from this simple example that the model places disconnected vertices within a neutral distance from each other, while positively/negatively connected vertices are put closer/further w.r.t. the neutral distance.

In addition to the signed versions of LinLog and FlexGD, we have built signed versions of some well-known models, namely Spring Electrical and Davidson and Harel, through the dual-primary method. The resulting models are shown in Table 3.4. $f(\|x_i - x_j\|)$ and $g(\|x_i - x_j\|)$ are obtained by integrating the force functions in the Spring Electrical and Davidson and Harel models. SE and dp stand for Spring Electrical and dual-primary, respectively.

dual-primary Spring Electrical	SE_{dp} =	$\sum_{\{i,j\} \in E^+} \frac{k_1}{3}\|x_i - x_j\|^3 - \sum_{\{i,j\} \in E^-} k_2 \ln \|x_i - x_j\|$
	+	$\sum_{\{i,j\} \in V^{(2)}} (\frac{1}{3}\|x_i - x_j\|^3 - \ln \|x_i - x_j\|)$
dual-primary Davidson and Harel	DH_{dp} =	$\sum_{\{i,j\} \in E^+} k_1 \|x_i - x_j\|^2 + \sum_{\{i,j\} \in E^-} \frac{k_2}{\|x_i - x_j\|^2}$
	+	$\sum_{\{i,j\} \in V^{(2)}} (\|x_i - x_j\|^2 - \frac{1}{\|x_i - x_j\|^2})$

Table 3.4: Signed energy functions.

Comparison of Signed Force-directed Models

Figure 3.25 shows the signed layouts of a graph containing 5 groups of vertices. Four groups out of five represent clusters (communities of friends for example). Each cluster is a clique. The last group consists of 15 isolated vertices with no link towards the rest of the graph (like hermits in social networks). Clusters 1, 2, 3 and 4 are represented with black, gray, blue and green, respectively. The disconnected vertices are white. There are two edges between clusters 1 and 2, indicating some common interests between them. There are no edges between cluster 4 and the rest of the graph, suggesting cluster 4 is an isolated society. clusters 1 and 2 dislike cluster 3. It is seen that FlexGD_s^{log} (Figure 3.25b and Figure 3.25e) has the best performance in representing the clusters. It is in particular more sensitive to the parameters, as increasing k_1 from 1 to 3 leads to more separation of clusters. Indeed, provided the vertices were discolored, and the edges were not presented, one could not identify the clusters using the primary-dual versions of SE and DH (Figures 3.25a, 3.25c, 3.25d and 3.25f). On the contrary, this is easily feasible in FlexGD_s^{log} . We also observe that disconnected vertices take position mostly in the frontiers of the FlexGD_s^{log} layouts, preventing them from being noisy to the rest of the graph.

3.8.3 Spectral Approaches for Drawing Signed Graphs

In its usual formulation, as discussed in Section 3.3.3 and in [80], the spectral approach can draw signed graphs, provided the condition of positive semi-definiteness of the Laplacian is satisfied. This condition is to make sure that the Hall energy function never becomes negative, that is, $\forall x \in \mathbb{R}^n : \sum_{\{i,j\}\in E} \omega_{ij}(x(i) - x(j))^2 \geq 0$. In the resulting layout, the vertices connected with negative edges would lie further from each other than those connected with positive edges.

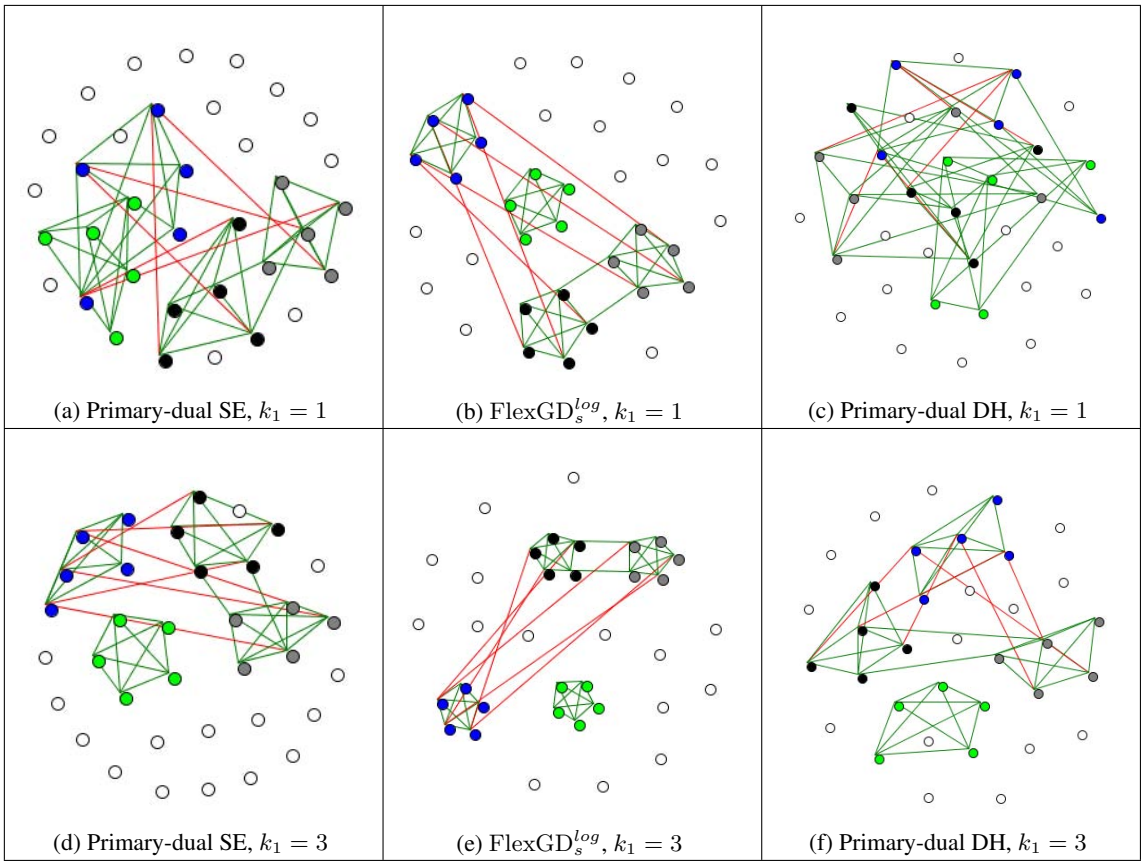
In [86], the authors present a different spectral approach for drawing signed graphs. They define the *signed Laplacian matrix* as $\bar{L} = \bar{D} - A$, where the *signed Degrees matrix* $\bar{D}$ is a diagonal matrix with non-zero entries defined as $\bar{D}_{ii} = \sum_{k\neq i} |\omega_{ik}|$. It is proved that the signed Laplacian is positive semi-definite in general. It is strictly positive definite, i.e. $\forall x \in \mathbb{R}^n : \sum_{\{i,j\}\in E} \omega_{ij}(x(i) - x(j))^2 > 0$, if and only if the signed graph is *unbalanced*. A signed graph is unbalanced, if it contains at least one cycle with an odd number of negative edges. The signed Laplacian can be used to draw signed graphs such that each vertex is positioned in the centroid of the vertices adjacent to it with positive edges and the opposite of those adjacent to it with negative edges. For vertex i, this can be formulated as:

$$x_i = \frac{\sum_{\{i,j\}\in E} \omega_{ij} x_j}{\sum_{j\in N(i)} |\omega_{ij}|}$$

In the matricial form, it can be written as:

$$\begin{aligned} \bar{D}x &= Ax, \\ \bar{L}x &= 0. \end{aligned}$$

The best solution to the above equation is the eigenvector of $\bar{L}$ corresponding to the largest non-zero eigenvalue. The next eigenvector can be used as the second dimension of the drawing.



(a) Primary-dual SE, $k_1 = 1$ (b) FlexGD$_s^{log}$, $k_1 = 1$ (c) Primary-dual DH, $k_1 = 1$

(d) Primary-dual SE, $k_1 = 3$ (e) FlexGD$_s^{log}$, $k_1 = 3$ (f) Primary-dual DH, $k_1 = 3$

Figure 3.25: Signed layouts of the toy example with 4 clusters and 15 disconnected vertices represented with white color. Cluster 1 is black, cluster 2 gray, cluster 3 blue and cluster 4 green (for all three models $k_2 = 1$). Positive edges are plot in green, and negative edges by red lines. FlexGD$_s^{log}$ has obvious superiority in the representation of clusters. Its clustering property is more sensitive to the parameters of the model.

3.9 Discussion

Force-directed models [40, 36, 68, 101, 25] are the most popular technique in graph drawing. The initial versions suffered from high running time. The Barnes and Hut algorithm [5] alleviated this problem. With the advent of multilevel algorithms [128, 51, 57], force-directed algorithms became rather efficient both from the computational and quality points of view. Another line of work which has recently become popular is spectral graph drawing [75, 19, 108], where the (generalized) eigenvectors of the Laplacian/Adjacency matrix of the graph used for drawing. Despite very low execution time, their layout quality is usually lower than force-directed algorithms.

In this chapter we suggested FlexGD and FlexSpec, a force-directed and a spectral approach, for drawing the graphs in different levels of structure abstraction. FlexGD fills optimally a circular drawing area. FlexGD is suitable for cluster visualization and extends this property of the LinLog model. In general, the layout is decent in the frontiers, and the symmetries are shown well. From an applicative point of view, we examined the model on graphs arising from a wide variety of real world applications like web graphs, 2D/3D problems, structural problems, electromagnetic

problems, social networks, etc. Although no single model can be claimed to have better performance on all graphs, as the suitability of a visualization model depends on the graph topology and the visualization requirements, it seems that in many cases FlexGD layouts are pleasing as much as, sometimes even more than, the layouts of the previous models. In addition, the user has more control over the layout with FlexGD than the previously considered models thanks to its abstraction parameter. We also tried other force shapes to build further flexible force-directed models with different properties. However, their layouts were massively inferior to those of LinLog. It seems that the relation between the logarithmic and linear functions (i.e. $\frac{f'(z)}{g'(z)} = z$) has a central role in moderate and controllable behavior of the FlexGD model.

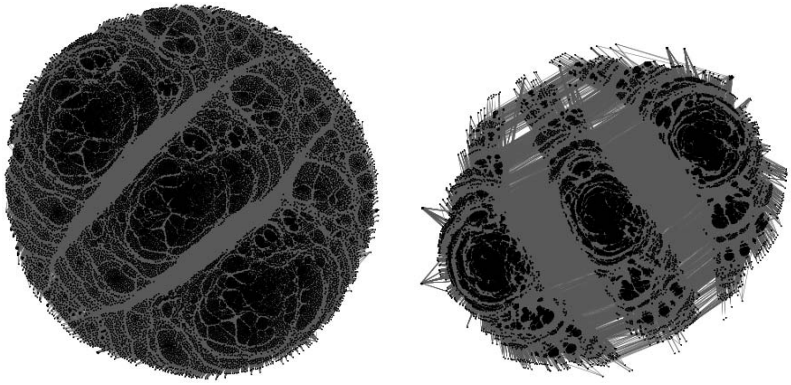
Nevertheless, FlexGD has the tendency to give a circular shape to the layouts. The circular layout is the natural legacy of using both attractive and repulsive forces to distribute the vertices over the layout. This design choice was originally made to give the abstraction capability to the model. If the width and the length of the graph structure are proportional, this is not a severe constraint, specifically because the circular shape of the layout decreases with k. Fortunately, the majority of real-world applications give rise to such graphs. However, in fewer cases where the length and the width of the graph are very disproportional (like the line graph in Figure 3.1b), the circular shape of the layout may disturb the graph structure by enforcing it to be packed into a disc. The FlexGD layout of the bcsstk31 graph in the first row of Figure 3.20 is an example of this issue. Though, one should note that very long-shaped structures (like a 20×400 grid) are known to be one of the most challenging types of topology for all models of graph drawing.

Most previous works adopt an empirical approach to validate their model. A distinguishing point of this work is to adopt a more formal approach initiated by Noack [101]. This was pretty helpful, as guided us through sophisticated parameterization of the force algorithm such that it copes well with the multilevel algorithm acting in its bottom. Furthermore, it gives a more quantitative understanding of the behavior of the model and clarifies its differences from the LinLog model.

The FlexGD model gives a unique perspective into the community (cluster) structure of huge sparse graphs rising from domains like Web applications. Examples of such layouts were shown in Figures 3.12 and 3.13. We believe for such graphs, drawing in the goal of visualizing the community structure is more indicative than using the conventional drawing criteria, as this latter usually results in a hardly informative clutter of interconnected vertices.

Apart from its direct links to the LinLog model, FlexGD has also similarities with the Binary Stress Model of Koren and Çivril [81]. This latter also has the property of abstractability. However, apart from its major differences from FlexGD both in the energy function and in the optimization technique, as posited by Theorem 3.4.1 and Theorem 3.4.2, linear attraction and linear-logarithmic pairwise energy functions of FlexGD make it more efficient in uniform vertex distribution and in cluster visualization. For example, the frontiers of the Binary Stress layouts are denser than the center. The authors add a random perturbation improving occasionally this drawback. Such a problem does not arise with FlexGD, as the linear-logarithmic shape of the pairwise forces results in perfectly even distribution of the vertices over the drawing area (see Figure 3.1).

Heavy Edge Collapsing (HEC) and Multiple Independent Vertex Set (MIVS) coarsening strategies were applied in the multilevel minimization algorithm. For some topologies, specially those containing star-shaped components, these coarsening strategies are ineffective, as the number of vertices of the coarser graph remains very close to that of the finer graph. Hence, developing more robust coarsening schemes may be considered as a direction for the future work. This problem is not particular to FlexGD, and has been reported by previous authors too [57]. Nevertheless, FlexGD



(a) FlexGD model, $k = 200$ (b) FlexGD model, $k = 1000$

Figure 3.26: gupta1 graph

can reveal the structure of some clustered graphs much better than the other models without needing any more advanced multilevel schemes. An example is the Gupta/gupta1 graph from the University of Florida Sparse Matrix Collection. In [26], the authors were forced to design a coarsening scheme specifically for this graph to reveal the three groups of vertices. Interestingly, FlexGD reveals these clusters easily without any coarsening phase. Figure 3.26 shows the FlexGD layout of this graph in two levels of abstraction. The multilevel algorithm suggested for the minimization of the FlexGD energy function is applicable to the LinLog energy function too. This is of importance, specially because in [101] the author only presents the LinLog energy function, but suggests no scalable and efficient algorithm for its minimization.

We only treated the case of unweighted graphs. Though, the model can be easily generalized for weighted graphs by integrating the edge weights into the attraction term of the energy function. This gives the following equation for the FlexGD energy function of weighted graphs:

$$\mathrm{FlexGD}(\mathbf{X}, A, k) = \sum_{\{i,j\}\in E} k\omega_{ij}\|x_i - x_j\| + \sum_{\{i,j\}\in V^{(2)}} \|x_i - x_j\| - \ln\|x_i - x_j\|,$$

where ω_{ij} is the edge weight between i and j, and A the adjacency matrix containing the edge weights.

FlexSpec was suggested as a spectral approach for flexible graph drawing. The main difference of FlexSpec with usual spectral graph drawing is that FlexSpec places each vertex in the weighted centroid of *all* other vertices biased towards its neighbors, while in the usual approach each vertex is placed approximately in the center of its neighbors. For high values of the abstraction parameter k, FlexSpec layouts are as good as the drawings obtained from the degree-normalized eigenvectors. The main advantage of these two approaches over graph drawing using the non-generalized eigenvectors is their preventing the low degree vertices from being isolated from the rest of the graph. Nevertheless, the behavior of FlexSpec is different from FlexGD, as this latter allows for more efficient control of the edge length with respect to the mean distance between vertices, and has different clustering properties.

Finally, drawing of signed graphs with force-directed models and the spectral approach was discussed. It was shown that both linear and logarithmic repulsive energies may be used in

FlexGD to set the negatively connected vertices excessively apart from each other. However, they result in different formulations for the equivalent minimization problem and the clustering of bipartitions. Namely, the signed version of FlexGD with logarithmic repulsive energy, i.e. $\mathrm{FlexGD}_s^{log}(\mathbf{X}, k_1, k_2)$, captures the ratio of k_1 times the length of positive edges to k_2 times the length of negative edges, while the signed version of FlexGD with linear repulsive energy, i.e. $\mathrm{FlexGD}_s^{lin}(\mathbf{X}, k_1, k_2)$, captures their difference. We also showed how a dual-primary approach may be applied to build signed energy models from the existing unsigned models.

CHAPTER 4

DATA VISUALIZATION VIA COLLABORATIVE FILTERING

Abstract

In this chapter, we suggest methods for global and personalized visualization of CF data. Users and items are first embedded into a high-dimensional latent feature space according to a predictor function particularly designated to conform with visualization requirements. The data is then projected into the 2-dimensional space by Principal Component Analysis (PCA) and Curvilinear Component Analysis (CCA). Each projection technique targets a different application, and has its own advantages. PCA places *all* items on a *Global Item Map (GIM)* such that the correlation between their latent features is revealed optimally. CCA draws *personalized Item Maps (PIMs)* representing a *small subset of items* to a specific user. Unlike in GIM, a user is present in PIM and items are placed closer or further to her based on their predicted ratings. The intra-item semantic correlations are inherited from the high-dimensional space as much as possible. The algorithms are tested on three versions of the MovieLens dataset and the Netflix dataset to show they combine good accuracy with satisfactory visual properties. We rely on a few examples to argue our methods can reveal links which are hard to be extracted, even if explicit item features are available. At the end, we discuss how matrix factorization can be used to devise new evaluation methods and more exact recommender algorithms.

4.1 Introduction

Following the Netflix Prize competition, most works on CF concentrated on improving the prediction precision [78, 90, 112]. Nevertheless, precision is not the only determining factor for the success of RSs. It is also important that results are represented nice and informative, as their careless presentation wastes efforts made to enhance precision. Most often, recommendations are presented in the form of a ranked list. Users have multiple interests, and are likely to change their preferences over the time [130, 78, 73]. In the same way, one user account may delegate several persons, each having different taste and needs. Though, ranked lists do not unveil the correlation between items unless the user knows them a priori. This issue limits the benefit of recommender systems, as it prevents users from making a correct selection based on their current needs. For example, a Netflix user may watch documentary movies during the week, but prefers action comedies for the weekend. She also uses the same account to search cartoons for her children. As a result, her recommendation list contains a mixture of different genres. However, it can not be understood from the list which movie is similar to the nice action movie she watched over the last weekend or is the same as the cartoon her children enjoyed last week.

In this chapter, *we suggest a Matrix Factorization-like (MF) approach to visualize the results of CF in the form of 2-dimensional maps*. This latter is a better alternative to ranked lists as it arranges items based on their similarity. Of course, other solutions like showing the content information of items may also be considered to improve the informativity of recommendations. However, it requires expensive gathering and involved cleaning tasks which companies try to avoid. In addition, interpretation of a visual map is more convenient and quite intuitive. There is an adage saying "a picture is worth a thousand words".

MF RSs [34, 8, 122] embed users and items in a latent feature space. Missing ratings are then estimated through the inner product of user and item coordinates. MF has been noted on passing to have the property of putting similar items close to each other in the latent feature space [79]. Nevertheless, there are two barriers against their application to draw 2-dimensional maps. First of all, the existing predictor functions are not apt for visualization purposes. Namely, users link intuitively the relevance between users and items to their Euclidean distance on the map. However, the existing predictors, generally variants of the inner product of vectors, need other information (e.x. the angle between user and item coordinates vector). Secondly, MF approaches need more than 2 dimensions to reach their optimal precision. Consequently, proper projection methods must be picked up preserving as much information as possible.

Our approach is composed of two phases: the embedding phase and the projection phase. The first phase embeds users and items into a high-dimensional space based on a predictor aimed at meeting visualization requirements. This predictor is a decreasing function of the Euclidean distance between the corresponding user and item. In the second phase, data is projected from the high-dimensional space into a 2-dimensional map. Two types of maps are presented, each aimed at a different goal. A *global Item Map (GIM)* is laid out using Principal Component Analysis (PCA). GIM shows *the whole set of items* in the 2-dimensional space capturing the most variance of the data from the original embedding space. It is advantageous to view how items are grouped according to the overall opinion of users. Independently of the underlying application, it is a general solution to the visualization of CF data. In order to represent a *limited number of items* to a specific user, we suggest *personalized Item Maps (PIMs)*. They are interesting in applications like visual recommending. PIMs place items around the active user such that their relevance decreases with their

distance from her. They prefer to preserve the *local* structure of the high-dimensional space. In other words, their priority is to preserve short distances rather than long ones. PIMs are generated using Curvilinear Component Analysis (CCA). The original CCA approach is altered in order to give more importance to preservation of distances w.r.t the active user. All PIMs are presented with the same visual parameters such that comparison between PIMs of different users is meaningful.

The precision of our approach is validated running experiments on the three versions of the MovieLens dataset and the Netflix dataset. The concentration in this chapter is on recommending movies. Despite, the same approach can be used for any other type of data like musics, restaurants, books, etc. CF does not require content information of the items. Despite, we show with a few examples that our maps are very good in unveiling semantic correlations between items.

4.2 Background

Visualization of CF has been previously discussed in a few works. In [20] for example the *NEAR* panel is suggested to manage digital collections. A graphical interface is provided to show the latent similarities between items to the end user in a visual manner. This interface is a hierarchical browser, but provides no map. In [83], the authors combine users' listening behavior with social taggings to derive *social audio features* of the songs. Probabilistic Latent Semantic Analysis (PLSA) is consequently applied for dimensionality reduction. In the context of Neighborhood-based CF, classical Multi Dimensional Scaling (MDS) [23, 13] is directly applicable. MDS has been widely used as a visualization technique in many domains including data mining and machine learning. Namely, a symmetric matrix of item-item dissimilarities is computed. A user-specific list of top-k relevant items is then visualized through MDS. This method has the drawback of not mapping a user together with its recommended items, as similarity is only defined between items but not between a user and an item.

Another possible solution to CF visualization is to model the data as a bipartite graph with vertices representing users and items, and edges representing the interactions (e.x. [133]). The graph is then visualized by one of the graph drawing algorithms presented in Chapter 3. Though, graph drawing algorithms are aimed at drawing readable layouts according to a number of predetermined aesthetic criteria. Consequently, although they lead to visually pleasing results, they do not have convincing predicting ability. In particular, they offer no prediction function to estimate the missing ratings.

In this chapter, we restate the MF approach in the goal of targeting visualizability of the results, while preserving prediction precision. Relating the relevance of users and items to their distance paves the way for usage of dimensionality reduction techniques. Unlike Neighborhood-based visualization through MDS, our approach can project users and items together on the recommendation map. It is also worth noting the conceptual difference between Neighborhood-based and MF approaches. A study of the literature shows the two approaches treat the data in different scales [76]. Specifically, MF processes the whole data in a global scale. On the contrary, Neighborhood-based approach concentrates on local similarities inside data. This implies that MF maps and Neighborhood-based maps visualize the correlations inside the data in different scales.

Another asset of this work is aligning the properties of the projection approach with those of the predictor function. Namely, the more the items are relevant to the user, the hyperbolically closer to her they are placed by the predictor function. CCA then captures this property of the predictor by preferring the local topology over the global topology. As far as we know, there is no prior work adopting such a strategy to CF visualization. Our approach is also a solution to alleviate the

explainability issues of the MF approach. Visual representation of the MF data provides users with the possibility of choosing an item close to their past favorite items. It can be explained to the user that the other users think that the nearby items on the map have similar content and quality.

4.3 Matrix Factorization for Visual Prediction

Describing data with predefined features needs content information which in turn requires complicated gathering and cleaning tasks. One way to overcome this problem is to use dimensionality reduction techniques to describe the data by latent features. In machine learning, it is known as Latent Semantic Analysis (LSA). Singular Value Decomposition (SVD) is the most common MF approach to CF. The rating matrix is represented by R, where rows represent users and columns represent items. Each entry r_{ij} is the rating of user i for item j. SVD factors R in the form of $U\Sigma V^T$, where $U_{m\times m}$ and $V_{n\times n}$ are orthonormal matrices, and $\Sigma_{m\times n}$ is a diagonal matrix whose entries are the singular values of R. A low rank approximation matrix $\hat{R}$ is made by picking the largest singular values and corresponding left and right singular vectors. It is proved that SVD provides the best low rank estimation of R according to the Frobenious norm $\left\|R-\hat{R}\right\|_F$. In CF, it is convenient to reformulate the factorization in the form of $R = PQ^T$, where $P = U\Sigma^{-1/2}$ and $Q^T = \Sigma^{-1/2}V^T$. In this case the low rank estimation of P and Q denoted by $\hat{P}$ and $\hat{Q}$ contain user and item latent features.

4.3.1 Original SVD-like Approach

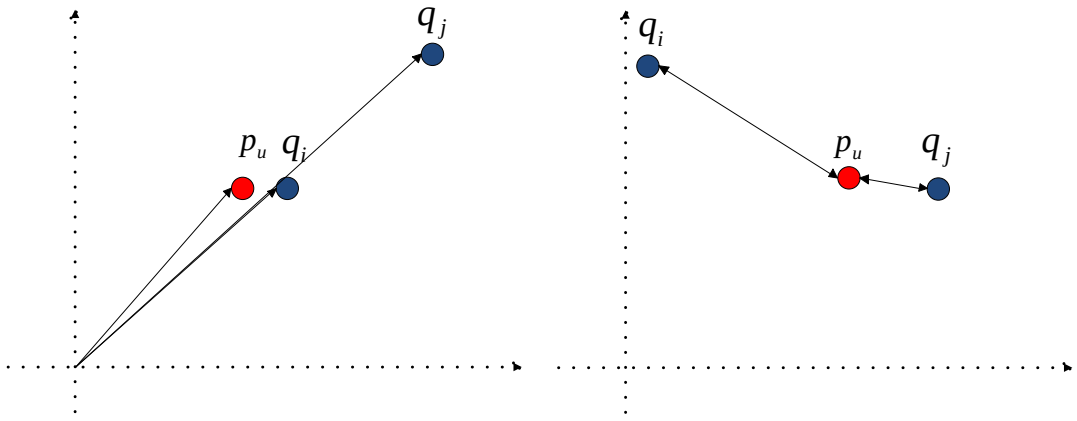
SVD is computable if all matrix entries are known. In CF however, the majority of ratings are missing. One way to deal with this problem is to fill in the missing entries by imputation. Nevertheless, inexact imputation can decrease the precision of recommendations. On the other hand, it significantly increases the complexity of the computations. A more successful alternative, known as the SVD-like approach, is to estimate the latent features through minimizing a regularized cost function.

$$min \sum_{p*,q*,r*\in R} (r_{ui} - p_u \cdot q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2),$$

where $p_u = (x_u(1), \cdots, x_u(k))$ and $q_i = (y_i(1), \cdots, y_i(k))$ are the latent features of user u and item i in an embedding space of k dimensions. The first term strives to minimize the prediction error over the observed ratings. Since the goal is to predict future ratings, the second terms is added to avoid overfitting the observed ratings. λ is called the regularization factor. This function is minimized using the stochastic gradient descent method. Users and items are moved iteratively by a step length γ in the opposite direction of the gradient until the movement of users and items is negligible. At each iteration, coordinates are updated as:

$$\begin{cases} p_u \leftarrow p_u + \gamma(e_{ui}p_u - \lambda q_i) \\ q_i \leftarrow q_i + \gamma(e_{ui}q_i - \lambda p_u) \end{cases},$$

where $e_{ui} = \hat{r}_{ui} - r_{ui}$ is the prediction error of item i for user u. A local minimum is usually found after few iterations (less than 100). λ and γ are optimized by cross validation on a smaller subset of the ratings.



(a) SVD-like predictor. j is more relevant to u than i, i.e. $p_u \cdot q_j > p_u \cdot q_i$, while it is further from her, i.e. $\|p_u - q_j\| > \|p_u - q_i\|$.

(b) Visual predictor. The more relevant item j is closer to the user u than i. The distance from the origin and the angle between the user and item coordinate vectors are not central.

Figure 4.1: The comparison between the suitability of the inner product and the Euclidean distance for visual prediction.

4.3.2 Basic Visual Predictor

The original SVD-like approach is not suitable for visualization purposes. The inner product function, used in the SVD-like approach, is a function of the magnitude of the vectors and the angle between them. Consequently, it can place a less relevant item closer to a user than a more relevant one. This presentation is misleading, as users intuitively interpret closer items as more relevant. More specifically, the inner product between the coordinates vector of a user, say p_u, and a far item, say q_j, may be a larger value than the inner product of p_u and the coordinates vector of a nearby item, say q_i. Such a problem does not arise, if the predictions are purely a function of the Euclidean distance, as the relevance between a user and an item is computed directly from their distance. Figure 4.1 illustrates why the inner product function does not conform with visualization requirements, while the visual predictor based on the Euclidean distance does.

Now, we come to the definition of the visual predictor. First, let us rewrite the regularized cost function in a general form. Let $\hat{r}_{ui} = f(\|p_u - q_i\|)$ be the predicted rating of user u for item i. The cost function has then the following form:

$$min \sum_{p*,q*,r*} (r_{ui} - \hat{r}_{ui})^2 + \lambda(\|p_u\|^2 + \|q_i\|^2).$$

Taking the partial derivatives, the coordinate update equations are obtained:

$$\begin{cases} x_u(j) \leftarrow x_u(j) - \gamma(e_{ui} \frac{x_u(j) - y_i(j)}{\|p_u - q_i\|} \frac{\partial \hat{r}_{ui}}{\partial x_j} + \lambda \frac{x_u(j)}{\|p_u\|}) \\ y_i(j) \leftarrow y_i(j) - \gamma(e_{ui} \frac{y_i(j) - x_u(j)}{\|p_u - q_i\|} \frac{\partial \hat{r_{ui}}}{\partial y_j} + \lambda \frac{y_i(j)}{\|q_i\|}) \end{cases}.$$

In the visual predictor, we set an inverse relation between the distance of an item from a user and the corresponding rating:

$$\|p_u - q_i\| = \alpha(\frac{1}{r_{ui}} - \beta).$$

The hyperbolic shape of the distance function is particularly useful in CF, as the more the items become irrelevant to a user, the faster they get further from her. This property helps us concentrate on visualizing local topology in PIMs while relaxing long distances. Taking the inverse function of the above equation, the objective estimation function is derived:

$$\hat{r}_{ui} = \frac{1}{\frac{1}{\alpha}\|p_u - q_i\| + \beta}. \tag{4.1}$$

In this case, the partial derivatives are:

$$\frac{\partial \hat{r}_{ui}}{\partial x_i} = \frac{-1}{\alpha \|p_u - q_i\| (\frac{1}{\alpha}\|p_u - q_i\| + \beta)^2}(x_i - y_i),$$
$$\frac{\partial \hat{r}_{ui}}{\partial y_i} = \frac{1}{\alpha \|p_u - q_i\| (\frac{1}{\alpha}\|p_u - q_i\| + \beta)^2}(x_i - y_i).$$

If we define

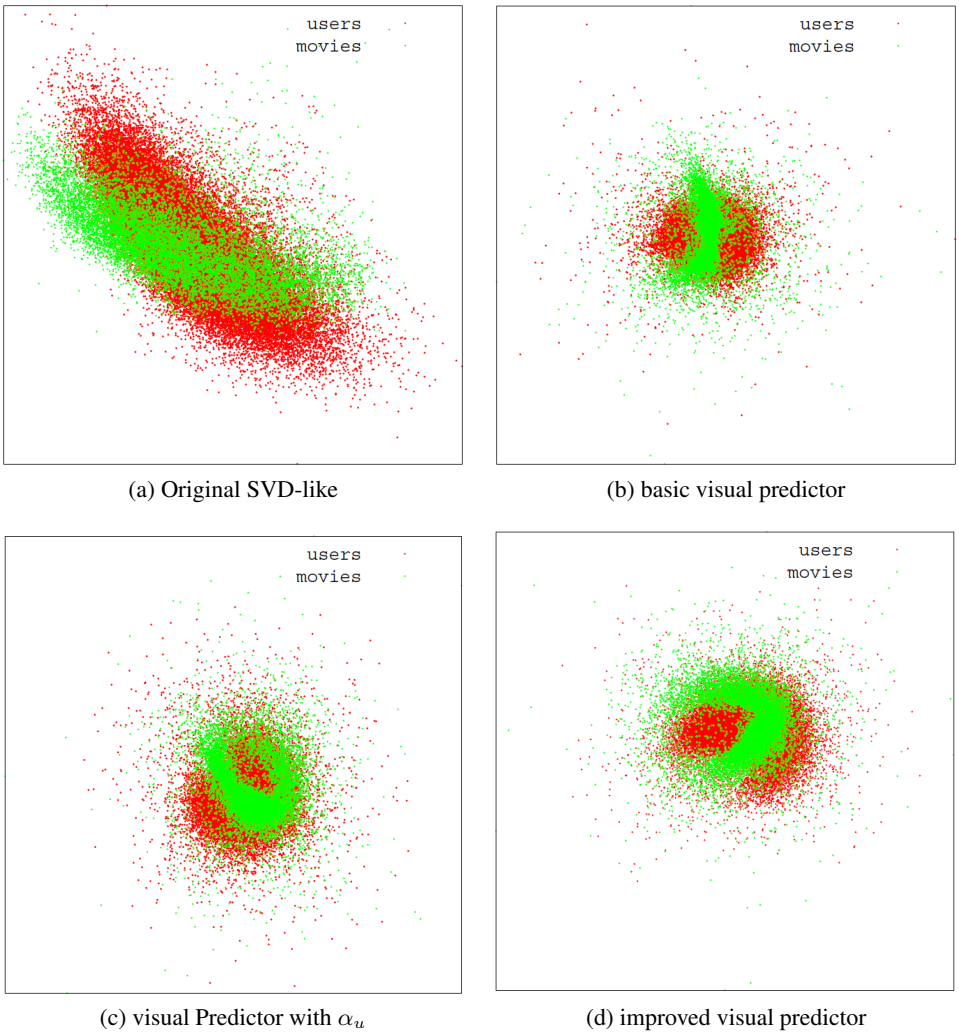
$$\triangle = \frac{1}{\alpha \|p_u - q_i\| (\frac{1}{\alpha}\|p_u - q_i\| + \beta)^2} = \frac{\hat{r}_{ui}^2}{\alpha \|p_u - q_i\|}$$

coordinates are updated at each iteration as:

$$\begin{cases} p_u \leftarrow p_u - \gamma(e_{ui}\triangle(q_i - p_u) + \lambda p_u) \\ q_i \leftarrow q_i - \gamma(e_{ui}\triangle(p_u - q_i) + \lambda q_i) \end{cases}, \quad \triangle = \frac{\hat{r}_{ui}^2}{\alpha \|p_u - q_i\|}.$$

α and β are found by cross validation. In theory, other functions varying inversely with the distance may also be used as predictor. We tried hyperbola square and tangent inverse functions. The precision was however worse than the presented predictor. This approach is very similar in the spirit to the MF approaches, as it embeds users and items in a feature space. Despite, since it is not formulated in form a factorization like SVD, we refer to it as an MF-like approach.

We compare different predictors on the MovieLens [46] and Netflix datasets [10]. To our best knowledge, these are the biggest datasets publically available. The algorithm is trained on 95% of the data. Predictions are made on the remaining 5%. The training and test sets are formed as in Chapter 2.5.1, that is, each user profile is split into 20 uniform regular slices. 19 slices are chosen as the training profile, while the remaining one serves as the test profile. The Netflix data set was made public for the Netflix Prize competition. The dataset includes $100,480,507$ ratings. $1,480,300$ of the same ratings were provided as the *probe set* for test purposes. We trained our algorithm on all ratings, after having discarded those included in the probe set. The predictions are made on the probe set. Both MovieLens and Netflix are based on a 5-star rating scheme. While the real ratings are integers, the predictions need not be. In our algorithm, predicted ratings are real numbers. Table 4.1 summarizes some properties of the four datasets. For both MoviesLens and Netflix datasets $\alpha = 2.5$ and $\beta = 0.2$ led to good results. For all datasets $\lambda = 0.01$. In the Netflix dataset, $\gamma = 0.005$ for the SVD-like approach and 0.00125 for the visual predictor. In the MovieLens datasets $\gamma = 0.002$ for the SVD-like approach, and 0.0005 for the visual predictor. Precision of the basic visual predictor is shown in Figure 4.3. It is seen that the precision improves with the number of dimensions up to some threshold. After that, the error increments because the data is not enough to train extra dimensions. In other words, the larger the dataset, the more dimensions can be trained. Figure 4.2 shows the distribution of MovieLens users and movies in the 2-dimensional space. It is obvious how much the distribution differs from one approach to the other.



(a) Original SVD-like

(b) basic visual predictor

(c) visual Predictor with α_u

(d) improved visual predictor

Figure 4.2: User and item distribution of different predictors in 2-dimensional latent feature space.

4.3.3 Enhancing the Basic Visual Predictor

Users have in general different rating habits to express their opinions about items. For example, they may rate movies with different personal scales, that is, one rates between 1 and 3, while the other rates between 3 and 5. In this case, the same score 3 shows like for the former but dislike for the latter. Consequently, the two users should not be considered similar even if they put the same notes for a number of common items. While a recommender system must be able to handle personal rating habits, the basic visual predictor cannot model it adequately. The reason is that it uses the same model parameters for all users. Figure 4.2b shows the distribution of users and items in a 2-dimensional embedding space using the basic visual predictor. It is seen that items are spread around a straight line with the end of the line slightly curved to the left. It seems that items can not be moved freely according to the needs of users. As a result, they are put along a line; Users then take some position around the area where the collection of their favorite items is denser. This issue

decreases considerably the precision of recommendations. In order to inject more flexibility into the model, we set α_u as a user-specific variable of the model learned automatically from the data. The prediction function is then:

$$\hat{r}_{ui} = \frac{1}{\frac{1}{\alpha_u}\|p_u - q_i\| + \beta}.$$

The regularized cost function is:

$$min \sum_{p*,q*,r*,\alpha*} (r_{ui} - \hat{r}_{ui})^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + \alpha_u^2).$$

The coordinates are updated as:

$$\begin{cases} p_u \leftarrow p_u - \gamma(e_{ui}\triangle(q_i - p_u) + \lambda p_u) \\ q_i \leftarrow q_i - \gamma(e_{ui}\triangle(p_u - q_i) + \lambda q_i) \\ \alpha_u \leftarrow \alpha_u + \gamma_\alpha(e_{ui}\triangle/\alpha_u - \lambda\alpha_u) \end{cases} , \triangle = \frac{\hat{r}_{ui}^2}{\alpha_u\|p_u - q_i\|}.$$

The values of λ, γ and β are the same as before. We observed that α must be moved with a bigger step length to improve the results. A proper value for all datasets is $\gamma_\alpha = 9\gamma$. The distribution of users and items is seen in Figure 4.2c. The pattern of items is more twisted this time (see Figure 4.2b) compared to the basic visual predictor, indicating they have been placed with more freedom. It is seen in Figure 4.3 that for all datasets precision is greatly improved compared to the basic visual predictor. For the small dataset of MovieLens100K, the visual predictor with user specific α slightly outperforms the SVD-like approach.

4.3.4 Improved Visual Predictor

The precision of the visual predictor with user-specific α is still worse than the SVD-like approach for larger datasets. To improve further the results, we also set β as a user-specific variable. This helps modeling user behaviors in finer levels, resulting in better positioning of users and items in the embedding space. The prediction function is:

$$\hat{r}_{ui} = \frac{1}{\frac{1}{\alpha_u}\|p_u - q_i\| + \beta_u}.$$

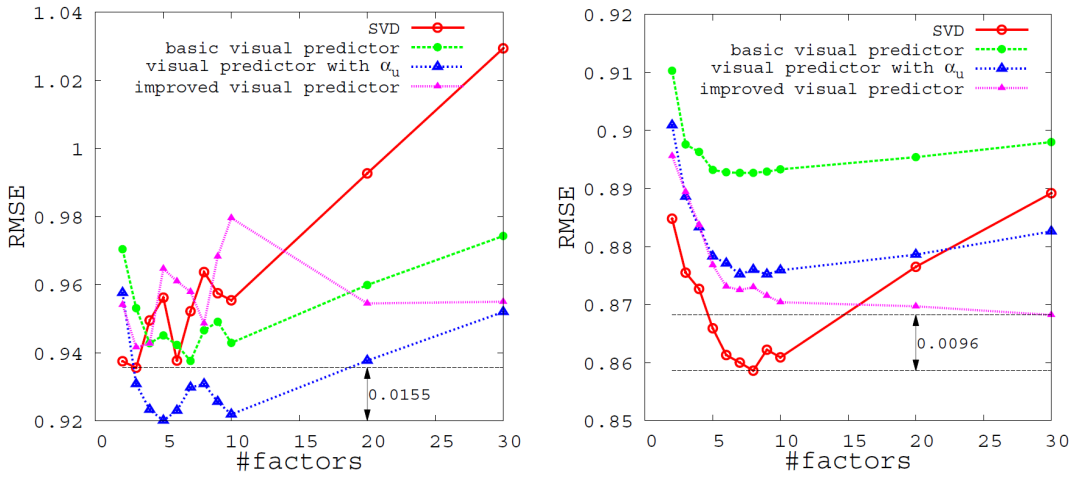
The regularized cost function is:

$$min \sum_{p*,q*,r*,\alpha*,\beta*} (r_{ui} - \hat{r}_{ui})^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + \alpha_u^2 + \beta_u^2).$$

The coordinates are updated using the following equations:

$$\begin{cases} p_u \leftarrow p_u - \gamma(e_{ui}\triangle(q_i - p_u) + \lambda p_u) \\ q_i \leftarrow q_i - \gamma(e_{ui}\triangle(p_u - q_i) + \lambda q_i) \\ \alpha_u \leftarrow \alpha_u + \gamma_\alpha(e_{ui}\triangle/\alpha_u - \lambda\alpha_u) \\ \beta_u \leftarrow \max\{\beta_u - \gamma_\beta(e_{ui}\hat{r}_{ui}^2 + \lambda\beta_u), \beta_u^{min}\} \end{cases} , \triangle = \frac{\hat{r}_{ui}^2}{\alpha_u\|p_u - q_i\|}.$$

The values of λ, γ, γ_α are the same as before, and $\gamma_\beta = \gamma/29$. We observed during the experiments



(a) MovieLens100K (b) MovieLens1M

(c) MovieLens (d) Netflix

Figure 4.3: RMSE of various approaches on different datasets.

that for a few users β_u becomes negative. In order to prevent it, we set a minimum value $\beta_u^{min} > 0$ for β_u. This is a necessary condition for unification of the scale of PIMs in Section 4.4.2. We set $\beta_u^{min} = 0.05$. The distribution of uses and items is shown in Figure 4.2d. The item pattern takes the shape of a horseshoe. It is wider and more symmetric than in Figure 4.2c. It is seen in Figure 4.3 that precision of the improved visual predictor is considerably better than the visual predictor with user specific α. For the small dataset of MovieLens100K, precision gets worse because the data is not enough to train the new variables. However, interestingly, it is almost as precise as the SVD-like approach for larger datasets. The difference in precision is always less than 0.01, being 0.0096, 0.0032 and 0.0057 for MovieLens1M, MovieLens and Netflix, respectively. This is a good sign as it allows for visual representation of the results without considerable loss in precision. Note it is still possible to improve slightly the precision adding item-specific parameters as in [78] or [72]. However, we avoid doing so because the predictions would not only depend on the active user and

Dataset	users	items	ratings	density%
MovieLens100K	943	1682	100000	6.3
MovieLens1M	6040	3883	1000209	4.24
MovieLens	69878	10681	10000054	1.33
Netflix	480189	17770	100480507	1.18

Table 4.1: Properties of the datasets

her distance from the corresponding item, but also on the properties of the item. Consequently, it is no more certain that items closer to a user have necessarily higher predicted ratings than those further from her.

4.4 Projection to Lower Dimensions

We embedded users and items into a high-dimensional space because the prediction error is less in higher dimensions. Nevertheless, the embedding is only half of the problem. Remember the goal was to provide item maps where the relevance between two items or that between a user and a movie can be conjectured visually through their Euclidean distance. Hence, we need proper projection methods to come back to 2 dimensions. We suggest two types of maps: *Global Item Map (GIM)* and *Personalized Item Map (PIM)*. GIM projects the whole set of items indicating the semantic correlation between them. PIM is aimed at representing a small set of items to a particular user such that their semantic correlation and predicted ratings conform to the Euclidean distance between them. In addition, predicted ratings depend inversely on the distance of the corresponding item from the active user.

4.4.1 Global Item Map

A Global Item Map (GIM) is very helpful to have an estimation of how items are similar or dissimilar according to consumers' point of view. It provides a way for clustering items without having any knowledge about their content. In the high-dimensional embedding space, each dimension is related to some latent feature of the data. The distance of users or items on each dimension is a measure of how similar they are in that feature. One way to visualize the data is to show only two features of the data [79]. However, the information of other features is missed. In addition, features are correlated and may contain redundant information. We use Principal Component Analysis (PCA) to generate a GIM. Data projection using PCA has backgrounds in Multi Dimensional Scaling (MDS) and graph drawing [52]. PCA finds the linear combination of all dimensions capturing as much variance of the data as possible. It omits linear dependencies between dimensions so that axes are uncorrelated in the output space. In other words, PCA represents the data from the most *informative* point of view. In CF, the two axes resulted from PCA are indeed two uncorrelated super features, giving the best explanation about the semantic correlation between items than any other linear combination of the original features.

Item latent features are kept in $\hat{Q}$. PCA uses the first eigenvectors of the Covariance matrix (those corresponding to the highest eigenvectors) as the principal components. To compute the principal

Algorithm 4.1 Power iteration algorithm

$\epsilon \leftarrow 0.0000001$ ▷ ϵ is some tolerance.
for i = 1 to k **do**
  $\hat{u}_i \leftarrow$ random
  $\hat{u}_i \leftarrow \frac{\hat{u}_i}{\|\hat{u}_i\|}$
  $u_i \leftarrow \hat{1}$
  while $u_i^T \hat{u}_i < 1 - \epsilon$ **do**
    $u_i \leftarrow \hat{u}_i$
    for j = 1 to i-1 **do** ▷ orthogonalize against previous eigenvectors.
      $u_i \leftarrow u_i - (u_i^T u_j) u_j$
    end for
    $\hat{u}_i \leftarrow S u_i$
    $\hat{u}_i \leftarrow \frac{\hat{u}}{\|\hat{u}\|}$ ▷ $\{\hat{u}_i\}$ are the estimated eigenvectors.
  end while
end for

components, $\hat{Q}$ is first centered:

$$\bar{\hat{Q}} = \begin{pmatrix} \bar{q}_1(1) & \cdots & \bar{q}_1(k) \\ \vdots & \ddots & \vdots \\ \bar{q}_n(1) & \cdots & \bar{q}_n(k) \end{pmatrix},$$

where $\bar{q}_i(j) = \hat{q}_i(j) - m_j$, and m_j is the mean of column j. Each column $\bar{q}_i$, representing one data point in the input space, is projected along the principal components:

$$y_i = \bar{q}_i^T u.$$

Writing it in the matricial form, the projection is defined as $Y = \hat{Q}^T u$. Since $\bar{\hat{Q}}$ is centered, Y is also centered. We need to maximize the Covariance of the projection:

$$\max (Y^T Y) = \max (u^T \bar{\hat{Q}} \bar{\hat{Q}}^T u) = \max (u^T S u),$$

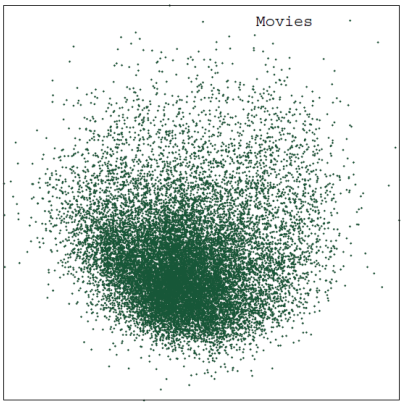


Figure 4.4: Netflix movies projected from 20d space by PCA

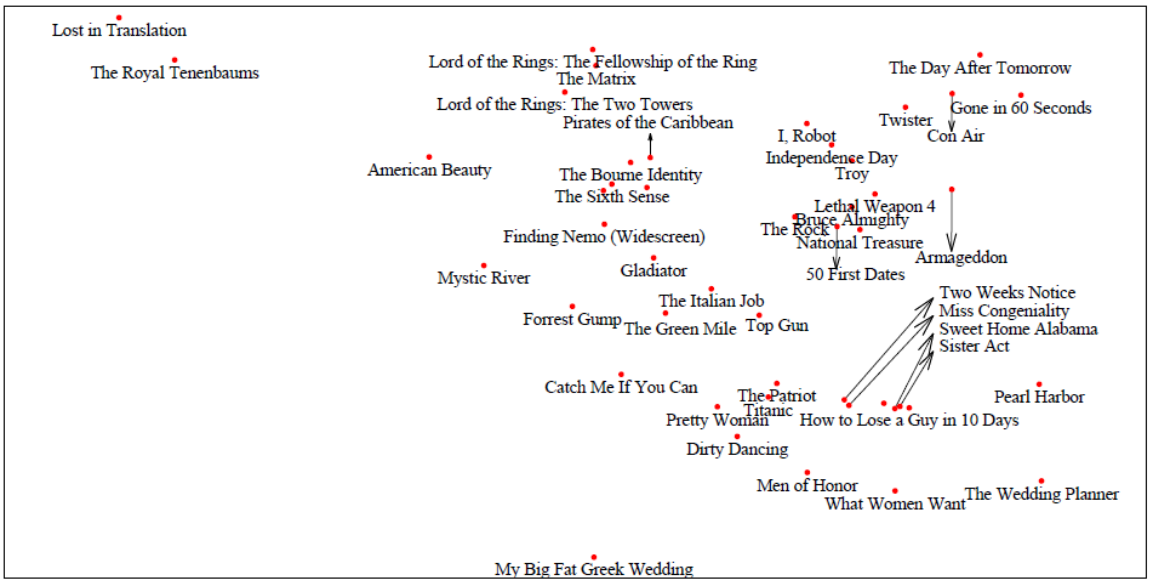


Figure 4.5: GIM of Netflix movies projected from 20-dimensional space.

where $S = \bar{\hat{Q}}\bar{\hat{Q}}^T$ is the Covariance matrix. This is a standard linear algebra maximization problem whose answer is the eigenvector of S corresponding to the highest eigenvalue. In the same way, the second eigenvector may be taken as the second dimension. The power iteration algorithm presented in Algorithm 4.1 can effectively compute the first eigenvectors. Since k is less than 50 in our application, the execution time is negligible.

Figure 4.4 shows the GIM of Netflix movies projected from a 20-dimensional space. It is seen that the distribution of movies is more uniform than in Figure 4.2d. This is an advantage of visualization as the drawing surface is used more optimally. Every subset of movies can consequently be picked up for the means of comparison. For example Figure 4.5 shows 47 movies with more than $140,000$ ratings in the Netflix dataset. Some names are omitted for clarity. With a little knowledge about the movies we see that the two very similar artsy movies *Lost in Translation* and *Royal Tenenbaums* are close to each other while they are far from *Pearl Harbor*, being a patriotic movie with elements of romance and drama. Also the two episodes of *Lord of The Rings* are close to each other. *The day after tomorrow* and *Twister* are both about natural disasters and are on top right of the map. Some continuous consistency of movie genre is seen on the map. Specifically, movies on the top left of the map are mostly artsy movies, while those on the top right are actions and adventures. These two groups smoothly join each other in the center just before meeting drama, romance and comedy movies on the bottom.

4.4.2 Personalized Item Map

Our second objective is to build Personalized Item Maps (PIMs). One possible solution is to take a desired set of items with a user and apply classical MDS (ex. [72]). If the data to be projected contains distances, which is the case in our application, classical MDS is equivalent to PCA. In CF, when all presented points are of the same type (for example movies), PCA positions pretty well similar points close to each other. This is intuitively understandable as each new feature recapitulates a

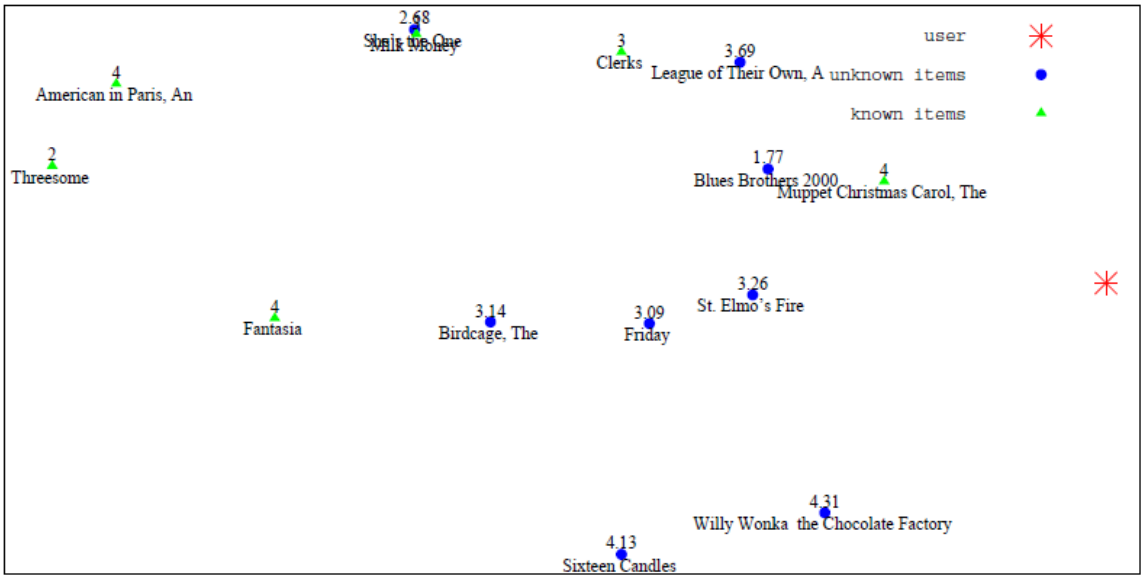


Figure 4.6: The item map of an anonymous MovieLens user projected using PCA from 10-dimensional space. The predicted ratings of unknown items is *not* a decreasing function of the Euclidean distance as we would like it to be.

combination of the semantics of the input features. Despite, when a set of items is to be represented to a user on a PIM, PCA disturbs so much the original distances from the active user that they no more conform to the predicted ratings. Figure 4.6 shows an example of a recommendation list for an anonymous MovieLens user. Shown in the figure, are the movies in her test profile together with a number of movies previously rated by the user. Scores of the known movies are the real ratings of the user. Scores of the test items are the predicted scores, computed in the high dimensional feature space. It is obvious that the distance between the unknown items and the user do not follow in general the hyperbolic relation that we had set during the embedding phase.

Indeed, PCA is a linear mapping preserving best the input distances, but giving the *same* weight to *all* distances. In order to increase the quality of PIMs to an acceptable level, we use non-linear mapping techniques to prioritize the preservation of shorter distances over longer ones. This idea is also in line with the hyperbolic relation we set in Section 4.3 between the predicted ratings and the Euclidean distance (between the corresponding user and item), as such relation emphasizes the closeness of relevant items.

Sammon mapping was the first variant of nonlinear maps addressing topology preservation [114]. However, it cannot reproduce all distances, because the output space has in general lower dimensions. Curvilinear Component Analysis (CCA) [30] was suggested as an alternative to Sammon mapping. The idea is to give priority to the preservation of *local* topology. This is done by starting from projecting long distances, and overriding them by shorter ones whenever compromises must be made.

We have slightly altered the original CCA approach to obtain better results in CF. The details are as follows. Suppose we want to show a subset of items together with an active user on a PIM such that first, the semantic correlation between items, represented by their distance, is inherited from the high dimensional embedding space. In the same way, items take distance from the active

user based on their irrelevance. Second, preservation of shorter distances is preferred over longer ones. Finally, preserving distances w.r.t the active user is more important as this latter is the central identity to whom the map is presented. We require the distances in the output space to conform with the original spaces as much as possible. The cost function to be minimized is then:

$$E = \frac{1}{2}\sum_{i}\sum_{j\neq i}(X_{ij} - Y_{ij})^2 F(Y_{ij}, \lambda_y),$$

where

$$F(Y_{ij}, \lambda_y) = \begin{cases} 1 \; if \; Y_{ij} \leq \lambda_y \\ 0 \; if \; Y_{ij} > \lambda_y \end{cases} .$$

X_{ij} and Y_{ij} are the distances in the original and the output space, respectively. i and j may refer either to an item or to the active user. We use the principal components as the initial distribution of the points. $F(Y_{ij}, \lambda_y)$ is the step function discarding all distances larger than λ_y in the output space. λy is decreased per cycle in order to prioritize shorter distances in the output space over longer ones.

The cost function can be minimized using the usual stochastic gradient descent approach. However, the complexity of each iteration cycle would be as $O(n^2)$. To decrease the complexity, a different procedure was proposed in [30]. Instead of moving every point according to the influence of all other points, one random point is fixed. Other points are moved with respect to the fixed point, but without consideration of the mutual influence between them. This approach reduces the complexity of each cycle to $O(n)$. The cost may occasionally increase in each cycle, but it decreases in average. If $F(Y_{ij}, \lambda_y)$ is the step function, the movement of each point j w.r.t the fixed point i is:

$$\Delta y_j(i) = \alpha(t) F(Y_{ij}, \lambda_y)(X_{ij} - Y_{ij})\frac{y_j - y_i}{Y_{ij}} \;\; \forall i \neq j.$$

λ_y is decreased per cycle, starting with λ_0 and ending with $\lambda_{t_{max}}$:

$$\lambda_y(t) = \lambda_0(\frac{\lambda_{t_{max}}}{\lambda_0})^{t/t_{max}}.$$

For each user we set $\lambda_{t_{max}} = \bar{d}_u$, where $\bar{d}_u = \alpha_u(\frac{1}{\bar{r}_u} - \beta_u)$ is the target distance corresponding to the average rating of user u. An item is relevant if its distance from the active user is less than $\bar{d}_u$. Setting $\lambda_{t_{max}} = \bar{d}_u$ ensures that items within some relevant distance from the user on the output space are not discarded until the end of the execution. In our experiments, we set $\lambda_0 = 2.0$. The algorithm is run a number of cycles on the active user and the selected items, denoted by Q_t. In order to give more weight to the preservation of the distances w.r.t the active user, whenever the fixed point is an item, the algorithm is run one extra time on the active user with probability 0.2. This strategy can be validated from a game theoretical point of view. Namely, each point of the data moves the others in the goal of aligning their position in the output space with their original distances from it. Since the algorithm is run in average $0.2\,|Q_t|$ times more on the active user, compromises are made to her benefit. Figure 4.8 shows the projection error of CCA for the same MovieLens user. Closeness to the 45° line indicates better consistency between the original and the output distances. It is seen that inconsistencies are less for shorter distances.

Application of user specific parameters to the visual predictor leads to different interpretation of distance in PIMs. It would be nice that all PIMs are represented in the same scale for users can

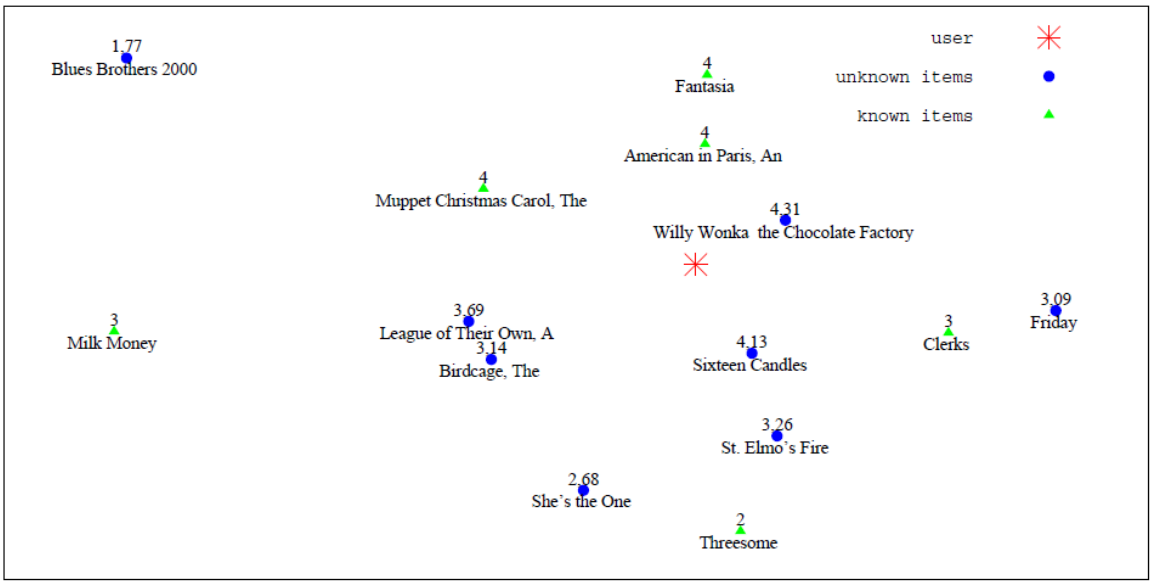


Figure 4.7: PIM of an anonymous MovieLens user projected from 10-dimensional space. The predicted ratings for items unknown to the user are in general a decreasing function of their distance from the user.

compare their PIMs with each other. To unify the scale of PIMs, we first set the origin to the position of the active user ($p_u = 0$), and reassign all the item coordinates in the translated coordinate system. We look for a transformation such that the predictions remain in the form of Equation (4.3.2) with the same α and β for all users:

$$\hat{r}_{ui}^{scaled} = \frac{1}{\frac{1}{\alpha}\left\|q_i^{scaled}\right\| + \beta}.$$

Setting $\hat{r}_{ui}^{unscaled} = \hat{r}_{ui}^{scaled}$, we obtain:

$$\left\|q_i^{scaled}\right\| = \frac{\alpha}{\alpha_u}\left\|q_i^{unscaled}\right\| + \alpha(\beta_u - \beta).$$

Since the Euclidean distance is non-negative, $\|q_i^{scaled}\| \geq 0$. This condition is satisfied for all users if $\beta \leq \beta_u^{min}$. The above equation is then held if:

$$q_i^{scaled} = \frac{\alpha}{\alpha_u} q_i^{unscaled} + \frac{q_i^{unscaled}}{\left\|q_i^{unscaled}\right\|}\alpha(\beta_u - \beta).$$

In our experiments we set $\alpha = 2.5$ and $\beta = \beta_u^{min} = 0.05$.

Figure 4.7 is the PIM of the same user. Note, that unlike in Figure 4.6the predicted rating is a decreasing function of the distance in the embedding space. It is seen that, in general, more relevant items are closer to the user than less relevant ones. The semantic correlation between items is preserved in a very good level. The movies *Friday* and *Clerks* on the right of the user are both comedies. In the same way, *Sixteen Candles*, *St. Elmo's Fire* and *ThreeSome* are movies targeting teenagers with a background of romance and comedy. *An American in Paris* and *Willy Wonka & the Chocolate Factory* are both musicals. This map can help the user choose a movie

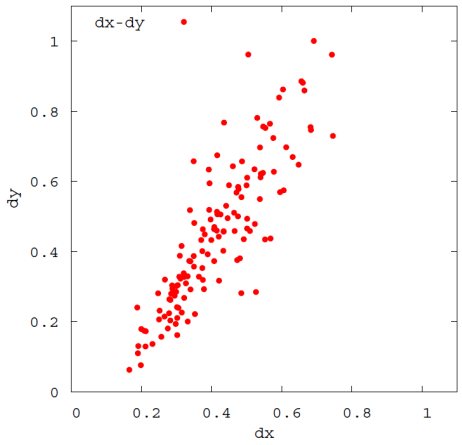


Figure 4.8: $d_x - d_y$ of the PIM of an anonymous MovieLens user.

similar to one of her previous favorites based on her mood in the time of selection. PIMs are also useful in offering *serendipitous* recommendations to a user. Serendipity addresses the diversity of recommended items. Serendipitous recommendations entice users to go through new experiences in their consuming behavior. If the user views a cluster of recommended items on her PIM, none of which is known to her, she is encouraged to go further and try them.

4.5 A Multi-interest Hybrid Recommender System

Both matrix factorization and neighborhood-based approaches have their advantages and drawbacks. The neighborhood-based approach enjoys good explainability and is intuitively easy to understand. On the other hand, matrix factorization methods are slightly more exact, but less explainable. The visual representation method discussed in this chapter is already an alleviation to their lack of explainability. Nevertheless, in this section we are going to introduce a hybrid recommender system that applies matrix factorization nicely in service of the neighborhood-based approach in order to enhance its precision and explainability.

Usually, the term *hybrid recommenders* is referred to techniques mixing different recommendation strategies like content-based and collaborative filtering. However, our hybrid recommender merges the two main schemes of the same strategy, i.e. collaborative filtering. Some works like [103] have already tried to estimate the similarity between items as the inner product of the coordinates vectors of items, computed through the SVD-like approach, in the phase of neighborhood formation of the item-based approach. Though, they report that this approach is not helpful for enhancing the precision. We have obtained the same results too. Instead, we take advantage of the clustering property of the matrix factorization approach to *weight* the contribution of each item/user in *the phase of similarity estimation* according to its relevance to the item/user for which the prediction is made.

The motivation behind this work is the drawbacks that can be associated with the well-established traditional similarity measures like Cosine or Pearson correlation. Although these similarity measures are widely applied, there is no theoretical support for their reliability. As a matter of fact, it seems that they fail to capture many important factors that must be naturally considered in esti-

mation of the similarity between users or items. One of these factors is the time. Namely, Cosine and Pearson similarities give the same weight to the old and the new ratings. Consequently, they have difficulties with detecting the change of users' interests or the fluctuations in the interest that items receive over a period of time. For instance, a user may have been interested in a special genre of movie for some time, but she is no more. Using the traditional similarity measures, the recommender system will continue to recommend such a user with the same genre of movie for long time after she is no more interested in that.

Another important drawback of these measures is their shortcoming in considering the multiple clusters of users or items. Specifically in the user-based approach, they consider naively the overlap between the previous ratings of two users without paying attention to the fact that the rated items may belong to different topics. Two users may agree on a topic, but have opposite opinions on an other. Likewise in the item-based approach, the traditional similarity measures give the same weight to the ratings of all users for the computation of the item-item similarities without considering how much the opinion of each user is similar to the user for whom the recommendations are computed.

One way to take into account multiple clusters of users/items is to use tags and proper tag clustering techniques to divide items/users into different topics/interest groups. In the user-based approach, each user is attributed with topic-specific neighborhoods of users having similar ideas to her in the corresponding topic. Equivalently in the item-based approach, each item is assigned with group-specific neighborhoods of items rated similarly by the users of the same interest group. This strategy results in a hybrid recommender system combining content-based and collaborative filtering strategies.

Our hybrid recommender algorithm is an alternative solution to capture the multi-cluster phenomenon in a purely collaborative filtering context where tags are not available. The central idea of this algorithm is to precompute the *latent* interest/topic similarity of users/items through MF, and compute the user-user or item-item similarities as a function of the item or user for which the recommendation is computed. This hybrid approach can come in item-based or user-based versions. The exact formulation and the intuitive interpretation for them are discussed in detail below.

4.5.1 User-based Hybrid Recommender Algorithm

Similar to its classic user-based counterpart, the hybrid user-based recommender estimates the similarity between two users considering their ratings for the items they have both rated. Nevertheless, the contribution of each rating is weighted with the relevance of the corresponding item to the item for which the prediction is made. As a consequence, the users whose ratings are closer to the active user about the items similar to the item for which the prediction is made receive a higher similarity score. This way, we avoid misinterpreting two users as similar, because they have close opinions about the items very different from the item of interest, while they may have opposite opinions about items similar to that. In order to estimate the relevance between two items we rely on the MF approach by applying the MF prediction function on their coordinate vectors.

If the rating of user u for item i is to be estimated using the k nearest neighbors, we need to find the k users who have firstly, rated i. Secondly, they have shown the most similar idea to u *about the items similar to* i. For an MF approach with prediction function f^{MF}, the *item-specific rating vector of a user*, say u, for item i is denoted by ν_u^i, where each element is defined as:

$$\nu_u^i(j) = \begin{cases} (r_{uj} - \bar{r}_u)\sqrt{f^{MF}(q_i, q_j)} \text{ if } j \in I_u \\ 0 \quad \text{otherwise} \end{cases},$$

where subtraction by $\bar{r}_u$ is to account for different rating scales of users. The *item-specific user-user similarity* between user u and user v for item i is denoted by $IUUS_{uv}(i)$. It is defined as:

$$IUUS_{uv}(i) = \log |I_u \cap I_v| Cos(\nu_u^i, \nu_v^i),$$

where $\log |I_u \cap I_v|$ is the term of significance weighting. Once the similarities between an active user u and all the other users are computed for item i, prediction estimation is done like in the classic user-based approach:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N(u,i)} IUUS_{uv}(i) * (r_{vi} - \bar{r}_v)}{\sum_{v \in N(u,i)} IUUS_{uv}(i)},$$

where $N(u, i)$ is the set of k users who have shown close rating behavior to u about the items similar to i, and have rated i too.

4.5.2 Item-based Hybrid Recommender Algorithm

While the user-based hybrid recommender computes the similarity between users, the item-based hybrid approach concentrates on the similarity between items. In this case, the similarity between items depends on the user for whom the predictions are computed. Unlike the classic item-based approach, the contribution of each user's rating in the similarity calculation is proportional to the latent similarity between her and the active user in the latent feature space. In other words, the item-based hybrid approach gives more importance to the ratings from tastewise users in the latent feature space. For an MF approach with prediction function f^{MF}, the *user-specific rating vector of an item*, say i, for user u is denoted by ξ_i^u, where each element is defined as:

$$\xi_i^u(v) = \begin{cases} (r_{vi} - \bar{r}_v)\sqrt{f^{MF}(p_u, p_v)} \text{ if } v \in U_i \\ 0 \quad \text{otherwise} \end{cases}.$$

The *user-specific item-item similarity* between item i and item j for user u is denoted by $UIIS_{ij}(u)$, and defined as:

$$IUUS_{ij}(u) = \log |U_i \cap U_j| Cos(\xi_i^u, \xi_j^u).$$

The predictions are then made as:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{j \in N(i,u)} IUUS_{ij}(u) * (r_{uj} - \bar{r}_u)}{\sum_{j \in N(i,u)} IUUS_{ij}(u)},$$

where $N(i, u)$ is the set of k items which have been rated most similarly to i by the tastewise users to u, and are also rated by u.

4.5.3 Complexity Issues

Unlike the neighborhood-based approach, where the similarity between two users/items is always a fixed value, the similarities in the hybrid recommender depend on the item/user for which the prediction is estimated. This means that the complexity of the hybrid approach is more than the plain neighborhood-based approach. Namely, we need to compute M^2N user-user similarities

instead of M^2 in the user-based approach, and N^2M item-item similarities instead of N^2 in the item-based CF. In order to decrease the complexity of the hybrid approach, users or items can be clustered in the latent feature space. The user-user/item-item similarities are then calculated for item/user clusters instead of for all items/users. Provided the users and items are clustered into C_U and C_I clusters, the complexity of the similarity computation will drop as M^2C_I and N^2C_U for the user-based and item-based approaches, respectively.

4.6 Discussion

We studied visual representation of CF data. Our approach establishes a link between semantic correlations within data points and their distance in the Euclidean space. Usually, visualization techniques are likely to sacrifice precision for readability. This work presents a unified approach capitalizing on high precision of the MF approach with user-friendly representation of the results. Relating the correlation of data points to their distance in a high-dimensional Euclidean space provides a framework for using the existing projection techniques. We defined our predictor with a hyperbolic function of distance to insist on similarities. Experiments on the three version of MovieLens dataset and the Netflix dataset shows that the visual predictor has almost the same precision as the inner product predictor. Interestingly enough, for the small MovieLens100K ratings dataset, the visual predictor even outperforms the SVD-like algorithm.

A Global Item Map (GIM) was generated using Principal Component Analysis (PCA). It can project the whole set of items at the same time. GIM has a satisfactory performance in regrouping similar items. An altered version of the Curvilinear Component Analysis (CCA) was used to generate Personalized Item Maps (PIMs). PIMs show a small set of items together with the active user. The main difference between PIM and GIM is that *local* topology is prioritized over global topology in PIMs. We also gave more weight to the preservation of distances w.r.t to the active user. All PIMs were reexpressed with the same prediction parameters for intra-user comparison purposes.

The running time of the MF approach is about half an hour on the Netflix dataset. Fortunately, the algorithm is run offline. It is recomputed with some frequency (for example daily or weekly). On the contrary, running times of projection algorithms is very short. Consequently, they both can be run on demand. However, there is no advantage in generating a GIM more than once after each execution of the MF algorithm as it is a global map.

The beauty of this approach is its capability in communicating latent information which are hard to understand even if one is disposed of the content information. Indeed, presented maps are the result of the collective wisdom of a huge number of users, sometimes being more promising than sniffing into tones of expert generated information. The application of the proposed techniques is not limited to recommendations. For example GIM can be used to monitor public opinion about products. Likewise, PIMs can be used to represent a visual comparison between the items of a *digital catalogue*. It is worth mentioning that although we discussed only item maps, the very same approach can be applied to users. For example a general user map can be generated to detect social groups of consumers showing the same habits in their consumption behavior. Another potential application of the suggested methods is to lay the foundation of an evaluation method for measuring the diversity of a recommendation list. Remember the similarity between items is a function of their Euclidean distance on the GIP. Furthermore, axes are uncorrelated and each contains new information about different features of items. Hence, the variance of the coordinates of the items in a recommendation list shows how diverse the recommendations are. If the recommendation list of

a user u is represented by $\hat{R}_u$, the diversity of this list is defined as:

$$\mathrm{var}(\hat{R}_u) = \sqrt{\frac{\sum_{\hat{r}_{ui} \in \hat{R}_u} (\|q_i - \bar{q}(u)\|)^2}{\left|\hat{R}_u\right|}}, \tag{4.2}$$

where $\bar{q}(u)$ is the centroid of the coordinates of the items in the recommendation list of u. In [87] the author defines the diversity as the fraction of items that change in the recommendation list of a user over the time. Our interpretation of diversity is however time-independent. Instead, it considers the subject diversity of recommendations. This can be an important asset to improve the quality of recommendation lists. Namely, If a user has shown interest in many genres of movies, it is more reasonable to squeeze out movies with high predictions but similar content in favor of more different movies. For instance, if a list of 5 recommendations are to be given to a user, there is no point to put the three versions of Star Wars, as are all similar material.

CHAPTER 5

CONCLUSION AND PERSPECTIVES

This chapter concludes the document by giving a summary of the previous chapters and potential directions for the future work. Since detailed concluding remarks have already been provided at the end of each chapter, we will mostly hint on open problems and ongoing projects.

5.1 Summary

In this document we studied the two topics of recommender systems and information visualization. Our study on recommender systems was focused on collaborative filtering (CF), as the most successful strategy to recommender systems. The two important approaches to collaborative filtering, the neighborhood-based technique and matrix factorization, were covered in Chapter 2 and Chapter 4, respectively. In Chapter 2, item-based and user-based approaches were compared in a P2P context. We applied epidemic protocols to propose a decentralized neighborhood formation scheme. The more important contribution of this chapter was a decentralized model-based algorithm based on random walks. The utility of this algorithm is in enhancing the precision of recommendations when the input data is very sparse. The effects of the factors influencing the behavior of the random walk algorithm, i.e. sparsity, neighborhood size and the similarity measure, were explained in detail.

In Chapter 2, we only discussed the decentralization of the neighborhood-based approach in a P2P context, but did not comment on the decentralization of the matrix factorization approach. Indeed, in a decentralized context with synchronous users, decentralization of the factorization-like approach is relatively straightforward. Namely, each user only needs to know her coordinates and the coordinates of the items she has rated. She then updates periodically these coordinates, the same way as in the centralized version, to decrease the prediction error per cycle. The only problem is to devise a collaborative mechanism to maintain and modify the coordinates of the items. This can be achieved either through a centralized public billboard where all users can read and modify the item coordinates, or by dividing the maintenance of item coordinates between all or some users. However, in a P2P context where synchrony may not be assumed, the decentralization of the factorization-like approach is more involved. If the maintenance of item coordinates is divided between users,

reallocation of items in case of churn is an important challenge. Furthermore, since users are not synchronous, those passing less time online will have less chance to modify the item coordinates with respect to their ratings. An advantage of the suggested decentralized neighborhood-based approaches is their capability in dynamically detecting and replacing the disconnected users, without affecting severely the performance of the recommender algorithm. This advantage rises from the the robustness of epidemic protocols against churn.

In Chapter 3, we provided a rather comprehensive review of the two major approaches to graph drawing, i.e. force-directed and spectral approaches. The goal of graph drawing is to glimpse a graphical understanding of the graph structure. The main contribution of this chapter was to suggest FlexGD, a Flexible algorithm for Graph Drawing. The important properties of FlexGD layouts are as follows. First, it can reveal the graph structure in different levels of abstraction through changing the ratio of the mean edge length to the mean distance between vertices. Second, it is efficient in cluster visualization. This property is inherited from the linear and logarithmic energy functions applied in the FlexGD energy function. Linear and logarithmic energy functions are shown in [101] to improve the visualization of clusters. Finally, FlexGD is capable of drawing disconnected graphs, as it is likely to distribute the vertices uniformly over the drawing area. Structure abstraction with spectral approach was also discussed in this chapter, and the sample layouts of the most important graph drawing algorithms were compared with the suggested models. In fact, apart from suggesting new drawing models for structure visualization, a strong point of this chapter is to explain and compare a large number of graph drawing algorithms with each other. We also examined drawing of signed graphs with force-directed and spectral approaches. The effect of the shape of the repulsive energy on the behavior of signed FlexGD was shown. Furthermore, we suggested a dual-primary approach for building signed energy functions of the existing unsigned models.

Chapter 4 suggests a novel method for efficient presentation of the recommendations to the users in the form of 2-dimensional maps. The key idea of this approach is to embed users and items in a latent feature space such that the pertinence of an item to a user is function of nothing but their Euclidean distance. This embedding space has usually more than 2 or 3 dimensions in order to offer recommendations with higher precision. The data is then projected back to 2 or 3 dimensions using multivariant techniques like Principal Component Analysis (PCA) and Curvilinear Component Analysis (CCA). The output of PCA is a Global Item Map (GIM), where similar items are placed close to each other. CCA generates Personalized Item Maps (PIM), where a specific user is represented with a limited number of items such that the relevance between her and the items decreases with their distance. This approach is a way to consider the final presentation of the recommendations at the same time that they are computed. Indeed, using this approach, the users can *watch* the logic the recommender algorithm has used to generate their recommendations. We also showed how the GIM can be used to devise new evaluation methods to measure the diversity of recommendation lists. An important advantage of the suggested scheme is that it is a purely collaborative filtering approach, needing no content information about the items or users.

The diversity of subjects and techniques applied in recommender systems and information visualization is larger than one can cover completely in a Ph.D. work. Hence, there are still ongoing projects and perspectives which are left to the future work. An ongoing project is an active recommender algorithm whose goal is to detect user preferences about movies by posing them a number of comparative questions. We briefly present this work which is in collaboration with Roger Wat-

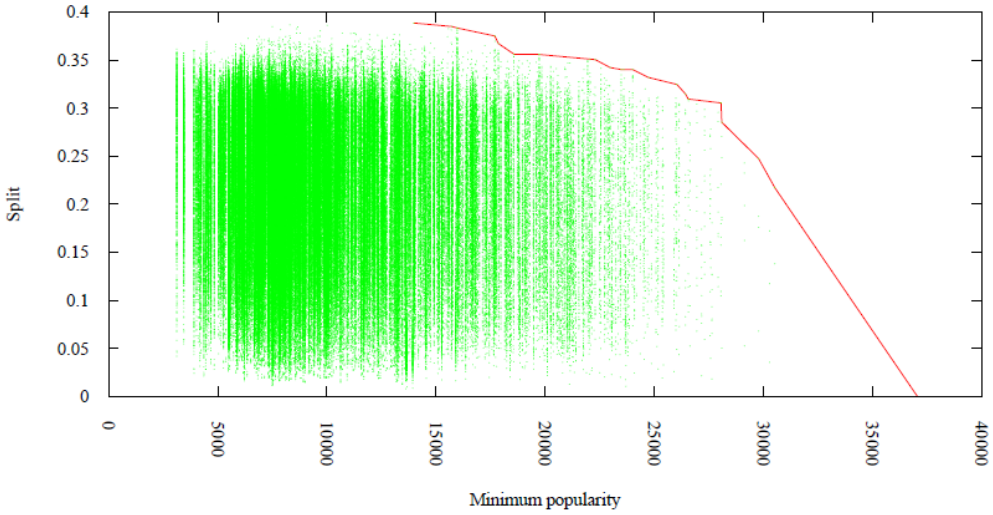


Figure 5.1: Parto frontier shown in red.

tenhofer (ETH Zurich), before commenting on the open problems in Section 5.3.

5.2 Active Learning for Movie Recommending

One of the well-known barriers against recommender algorithms is the problem of cold start. On one side, it is difficult to give satisfactory recommendations to new users that few in known about their preferences. On the other side, the initial experience of the users has a central role in their being convinced to continue using the recommender algorithm in the future. As a consequence, designers of recommender systems must always think of methods to fight against the lack of information in the phase of cold start. Data mining solutions such as the random walk algorithm suggested in Section 2 can be helpful to some extent. Though, most service providers prefer to build an initial profile for the users by asking them a few questions about their preferences when they register. A similar idea can be used to offer recommendations to unknown users once they have answered to some questions about their preferences. This strategy is known as *active learning in recommender systems* [24, 110, 121]. The main challenge of active learning is to provide satisfactory recommendations by asking as few questions as possible, similarly to the well-known 10 questions game. Otherwise, many users would be discouraged from finishing the registration process, or drop using the recommender algorithm.

We have implemented an active recommender algorithm for suggesting movies to users, once they have compared a number of movies with each other. Psychologists believe that human-beings are more performant in comparing two products than expressing themselves about each one individually. The algorithm relies on the Netflix Prize dataset to train a binary tree where every node represents a question about a pair of movies. In addition to the movie pair, each node is also associated with a set of users and a set of items of the dataset. The online user starts by comparing the two movies at the root, and goes through a path of the tree based on her preferences. At each node, the set of users or/and movies are pruned and pushed forward to the child nodes. The goal is that

the set of users, remaining in each node, gets increasingly similar to the online user. Likewise, the set of remaining items must be more and more representatives of her preferences. At the end of the tree, or even sooner provided she decides to give up, a sorted list of recommendations is offered to the user.

The first important issue is to determine a recommendation objective. For the moment we have worked on two objectives. The first one is to offer a best movie to the user. The second one is to collect as much diverse data as possible about the preferences of the user about different types of movies.

Once the recommendation objective is fixed, the rest of the problem is summarized in four questions:

1. How to find a pair of movies that its comparison by the user is likely to bear the most information for the chosen objective.

2. How to compute a subset of *users* of a parent node that must be pushed forward to a child node.

3. How to compute a subset of *items* of a parent node that must be sent to a child node.

4. How to generate the recommendations at each node.

The two versions of the algorithm with the above objectives are explained below in more detail.

5.2.1 Finding a Best Movie

The recommendation objective in this version is to help the user find a movie going best with her taste. At each node v, the online user compares the two movies forming the question of that node, say (i_l^v, i_r^v), where l and r stand for left and right, respectively. The root node contains all users of the dataset. Users having preferred i_l^v over i_r^v, that is, $r_{ui_l} > r_{ui_r}$ are sent to the left child and vice versa. In order to increase the chance of users knowing a movie, and to decrease the complexity of the computations, we fix one side of the question to the movie the user has voted for in the previous level. The reason is that if a user has preferred a movie in the previous level she knows it for sure. The user is then asked to compare the movie she has voted for in the previous level, say i_1, with the movie that the majority of users, having also preferred i_1, like more than i_1. In this approach, all nodes of the tree contain the whole set of items. At each node v, the prediction for each item i, represented by $\hat{r}_{vi}$, is estimated as the average of the ratings of the remaining users. In order to prevent items with very few but high ratings to come up in the recommendation list, we sort the predictions according to $\sqrt{N_i^v} \cdot \hat{r}_{vi}$, where N_i^v is the number of users in node v having rated i.

5.2.2 Detecting User Preferences

The recommendation strategy in this version is to explore the taste of the online user about different types of movies. The pair of movies in each question must be both popular and different. The popularity is for decreasing the probability of users' skipping the question. The two movies must also be different to help with understanding which genre of movie the user is interested in. Similar to the previous objective, we fix one side of the pair to the movie the user has voted for in the previous level. In order to find a pair of movies with the above properties, we choose from among those

which are *Pareto efficient* with respect to *popularity* and *split*. In the terminology of Economics, in a Pareto efficient system only allocations of goods are permitted that make at least one individual worse off. The set of choices which are Pareto efficient are called the Pareto frontier. In our case, the pairs of movies which have either higher popularity or higher split than any other pair form the Pareto frontier. An example of the split/popularity Pareto frontier is shown in Figure 5.1. The popularity of each movie in a node is an increasing function of the number of users of that node having rated it. We have tried the number of ratings, its square root and its logarithm to compute the popularity. The popularity of a pair is defined as the minimum of the popularities of the two movies. Therefore, if a pair has high popularity both movies of that pair are popular. At each node, the split of a movie i_1 w.r.t movie i_2 is the fraction of users in that node having preferred i_1 to i_2 over the number of users having rated both. The splits of two movies are dependent on each other. For instance, if there are no two users having put the same rating for the two movies, the splits of the two movies sum to 1. In general, the sum of the splits is always less than or equal to 1. The split of a pair is defined as the minimum of the splits of the two corresponding movies. Hence, the maximum split of a pair is 0.5 corresponding to when half of the users have preferred i_1 to i_2, while the other half have preferred i_2 to i_1. From among the pairs on the Pareto frontier, we choose the one maximizing $popularity * split$ as the question of the corresponding node. The prediction for each item is computed as the average of the ratings it has received from the users of the corresponding node. The ranking is done as in the previous version. In the simplest form of this approach, all nodes contain all items.

Pruning users based on few questions is prone to omit important data, as users have in general multiple interests which does not allow for such fast estimation of similarity or dissimilarity between users. As a consequence, an alternative approach was applied based on pruning the items. The properties of the items are more stable than those of the users. In order to prune the items, the SVD-like recommender algorithm was used. A fraction of movies, similar to the movie the user has not liked in the previous question, is omitted at each edge of the tree.

The project is still ongoing to implement further strategies and improve the quality of recommendations as much as possible. Future extensions include information theoretical approaches defining an *Entropy* for each question in order to ask the question with maximum entropy of information. Also, we are looking into devising evaluation methods to compare binary trees with each other. The current interface of our active movie recommender is shown is Figure 5.2.

5.3 Other Open Problems

Some other directions for potential future work are mentioned below.

Context-aware recommendations [118, 135, 29, 131] are probably the most important open problem in the domain of recommendation systems. Context-aware recommenders are aimed at customizing the recommendations for the special situation where they are to be presented to the user. Examples of context information include the time when the recommendation is given or the geographic location of the user. For instance, the location of the mobile users may be known to the recommender system thanks to the GPS. Therefore, if a user is on a trip it is more reasonable to recommend her the restaurants and commerces in her travel destination rather than those in her common living area.

Figure 5.2: The interface of the active movie recommender algorithm.

Self and multi-interest characterization of users The rationale behind collaborative filtering is that if a user has been in agreement with some other users in the past, she will agree with them in the future too. This axiom has of course many shortcomings in modeling the reality, as users may change their habits, needs and taste over the time. One perspective for future work is to develop algorithms to mine the pattern of user's behavior. Collaborative filtering uses the similarity of a user to the other users to provide her with recommendations. However, an important part of the information about each user resides in the pattern of her own behavior. It is therefore important to develop techniques for predicting the future behavior of a user based on her own history. In addition, users have multiple interest domains. Hence, they can agree with a group of users in one domain, but disagree with them in another. The multi-interest hybrid recommender system suggested in Chapter 4.5 is a solution to alleviate this problem. More robust investigation of this approach remains for future work.

Temporal effects Mining the temporal effects in RSs has recently drawn a lot of attention. Time-dependent matrix factorization methods like [78] or graph-based approaches like [132] have been proposed to deal with this problem. Nevertheless, none of these solutions report convincing enough improvement over non-temporal recommendations, and more works still need to be done in this context.

RESUMÉ EN FRANÇAIS

Nous avons assisté au développement rapide de la technologie de l'information au cours de la dernière décennie. D'une part, la capacité du traitement et du stockage des appareils numériques est en constante augmentation grâce aux progrès des méthodes de construction. D'autre part, l'interaction entre ces dispositifs puissants a été rendue possible grâce à la technologie de réseautage. En particulier, grace à l'Internet, les données, les services et les applications sont désormais accessibles de partout. Une conséquence naturelle de ces progrès, est que le volume des données générées dans différentes applications a grandi à un rythme sans précédent. En outre, de nouvelles perspectives ont été ouvertes sur la façon dont la technologie Web peut influencer notre mode de vie : avec l'avènement du Web 2.0, les utilisateurs peuvent désormais contribuer au contenu du Web. Cela a donné lieu à de nouvelles applications comme le réseautage social et le partage de contenu collaboratif. Avec plus de 900 millions d'utilisateurs, Facebook est toujours en croissance. Les utilisateurs de l'Internet partagent tout, allant des événements de la vie quotidienne aux services et applications, sur leur compte Facebook. De même, ils comptent sur des sites comme YouTube et Flickr pour diffuser leur contenu, exprimer leurs opinions et s'informer des opinions des autres.

Suite à ces étapes remarquables, nous sommes désormais confrontés à de nouveaux défis pour traiter et représenter efficacement la masse énorme de données à notre disposition. Bien que la disponibilité de l'information ne soit plus un problème, le volume de données est devenue tellement immense qu'il devient extrêmement difficile de trouver du contenu pertinent avec une qualité satisfaisante. Par conséquent, les utilisateurs ont besoin d'aide dans le processus de prise de décision pour trouver du contenu correspondant à leurs besoins. La tâche de l'exploitation des données permettant d'offrir du contenu personnalisé aux utilisateurs est connu sous le nom *recommandation*. Cependant, trouver du contenu pertinent ne représente qu'une partie du problème, car les résultats doivent également être présentés d'une manière informative. Une représentation efficace des données a beaucoup d'importance, à la fois pour accroître l'utilité de l'information aux utilisateurs, et pour développer des outils d'analyse des données efficaces.

Ce document est centré autour des deux axes de recommandaion du contenu pertinent et de sa visualisation correcte. Dans notre étude, nous allons nous concentrer sur le *filtrage collaboratif* comme l'approche la plus adaptée à des systèmes de recommandation. De même, nous nous concentrons sur le problème du *tracé de graphes*, comme la plupart des types de données peuvent être modélisés sous forme de graphes. Nous montrons aussi comment certaines des mêmes techniques utilisées dans les systèmes de recommandations peuvent être modifiées pour tenir compte des exigences de visualisation.

5.4 Décentralisation des Systèmes de Recommendations

Le problème de fouiller de l'information pertinente parmi la masse importante de données a créé un domaine de recherche populaire connu sous le nom *Systèmes de Recommendation* (SR). Plus précisément, les systèmes de recommandation traitent toutes des techniques du filtrage de l'information qui tentent de trouver des articles qui correspondent aux besoins d'un utilisateur, afin de lui fournir des *recommandations personnalisées*. Ces éléments peuvent être des pages Web, de matériel multimédia comme des films et de la musique, des nouvelles, des articles scientifiques, des livres, des forfaits touristiques, des restaurants ou tout autre objet sur le Web. La notion de personnalisation joue un rôle central dans la définition des systèmes de recommandation. Bien sûr, dans un scénario très simpliste, il est possible de calculer un classement général des articles, et de recommander les articles les plus populaires à tous les utilisateurs. Néanmoins, presque toute la littérature des SR est exclusivement consacrée aux recommandations personnalisées. Le domaine des SR est étroitement liée à l'apprentissage automatique (machine learning), la fouille de données (data mining) et la recherche de l'informations (information retrieval). En conséquence, de nombreuses techniques dans la conception des SR ont été adoptées à partir de ces domaines. Les systèmes de recommandation ont récemment connu une grande popularité à la fois dans l'industrie et dans le secteur de la recherche. Tous les principaux acteurs du marché du commerce electronique (ex. Amazon, eBay, YouTube, Netflix, MovieLens, Last.fm, Pandora, etc.) ont intégré des moteurs de recommandation dans leurs sites Web, et travaillent activement pour améliorer leur qualité. En effet, les systèmes de recommandation sont un atout redoutable pour donner satisfaction aux utilisateurs et pour la croissance de la clientèle. L'attention portée aux SR a considérablement augmenté à la suite de la compétition de Netflix Prize [61, 9, 99]. Le consours à $1M$ de Dollars avait pour but d'améliorer la précision de l'algorithme de recommandation existante de Netflix, appelé Cinematch, de 10%. Il a finalement été remporté par une coalition de trois groupes de recherche en Septembre 2009. Les gagnants ont publié un résumé de leur approche dans [77, 124, 106].

L'objectif principal d'un SR est d'estimer l'intérêt des utilisateurs pour les articles. Pourtant, les SR peuvent faire plus que la prédiction, à la fois pour les fournisseurs de services et pour les utilisateurs [17]. Les systèmes de recommandation peuvent également gagner la confiance des utilisateurs, quand ils leur fournissent des recommandations convaincantes. La confiance est un facteur important pour rendre les clients fidèles au site. Les données recueillies par un SR est une source précieuse d'informations pour l'analyse du marché et pour l'apprentissage des préférences des clients. En outre, les utilisateurs peuvent utiliser un SR pour devenir plus conscients de leurs préférences et/ou des articles existants dans le système pour leur prochaine consommation. Dans ce cas, le SR est exploité de la même façon qu'un moteur de recherche. Les utilisateurs peuvent également tirer parti des systèmes de recommandation pour exprimer leurs opinions, ou pour essayer d'influencer les autres utilisateurs.

La conception, la mise en oeuvre et l'évaluation des SR dépendent des exigences de l'application sous-jacente, le type de données à portée de main et les objectifs du SR. Certaines applications peuvent avoir des exigences particulières. Par exemple, dans certains contextes, un article ne peut être consommé plus d'une fois : un recommendeur de film ne doit plus recommander un film que l'utilisateur a déjà vu. Au contraire, un recommendeur de musique peut continuer à recommander le même article à plusieurs reprises, car les fans de musique ont l'habitude d'écouter leurs chansons préférées plus d'une fois.

Le *Filtrage Collaboratif (FC)* est la stratégie la plus populaire pour concevoir les systèmes de

recommandation. La raison de cette popularité est que le FC ne nécessite aucune information sur le contenu des articles, et a généralement une meilleure précision que les autres approches majeures des systèmes de recommandation comme l'approche basée sur le contenu (content-based recommenders). Parmi les différentes stratégies du FC, *factorisation de matrice* [76, 78] et *les modèles du voisinage* (neighborhood-based models) [54, 115] sont plus populaires que les autres. Les modèles du voisinage sont discutés en Chapitre 2, alors que l'approche de factorisation de matrice est mentionnée en détail en Chapitre 4. Bien que la factorisation de matrice surpasse légèrement l'approche fondée sur le voisinage en terme de la précision, ce dernier est l'approche la plus largement utilisée dans le FC en raison de certains de ses avantages comme d'être mieux explicable. Il est important pour un SR de pouvoir fournir la raison derrière ses recommendations. Dans l'approche fondée sur le voisinage, des explications peuvent être fournies sur la base des utilisateurs des articles similaires qui ont un rôle central dans le calcul des recommandations. Au contraire, les explications sont difficiles à donner pour le modèle de factorisation de matrice, parce que les prédictions sont faites en exploitant l'ensemble des informations dans le système, et un voisinage non-restreint.

Un modèle du voisinage se compose de deux phases : la formation du voisinage et l'estimation des notes. Dans la phase de la formation du voisinage, un ensemble d'articles similaires est formé pour chaque article (l'approche basée sur les articles (item-based approach)), ou bien un ensemble d'utilisateurs similaires est formé pour chaque utilisateur (l'approche basée sur les utilisateurs (user-based approach)) selon une mesure de similarité.

Dans la phase d'estimation des notes, le voisinage est utilisé avec une fonction de prédiction pour estimer les évaluations d'articles inconnus pour les utilisateurs. Presque tous les travaux dans les premiers jours du FC étaient basés sur les utilisateurs (user-based). Toutefois, l'approache basée sur les articles (item-based) a récemment reçu plus d'attention dans le domaine des algorithmes de recommandation centralisés pour sa meilleure "scalability". Plus précisément, dans la plupart des ensembles de données, le nombre d'utilisateurs est plus grand et croît plus rapidement que le nombre d'articles. L'approache basée sur les articles peut être mieux epliquée que celle basée sur les utilisateurs, car les utilisateurs ont une meilleure connaissance des articles que des utilisateurs.

Pourtant, des systèmes de recommandation sont confrontés à une quantité croissante de données à traiter, parce que le nombre d'utilisateurs en ligne augmente, ce qui nécessite généralement des coûteuses opérations de calcul et de stockage afin de fournir des résultats précis. Bien que cette combinaison de facteurs puisse saturer les systèmes centralisés, les approches entièrement décentralisées offrent une alternative intéressante avec de multiples avantages. Tout d'abord, le calcul des prédictions peut être réparti entre tous les utilisateurs, supprimant la nécessité d'un coûteux serveur central et améliorant le "scalability". Deuxièmement, un recommandeur décentralisé facilite la protection de la vie privée (le *privacy*) des utilisateurs, car il n'y a aucune entité centrale de stockage qui possède leurs informations privées. Plusieurs algorithmes existants tels que [18], peuvent être par conséquent déployés dans les environnements décentralisés afin de préserver la vie privée des utilisateurs en communiquant les opinions des utilisateurs sous une forme cryptée, sans divulguer leur identité. Enfin, un service de recommandation distribuée est un atout précieux pour les applications pair-à-pair (P2P) comme BitTorrent et Gnutella. Ces réseaux décentralisés sont devenus des médias très populaires de partage de contenu entre utilisateurs. Ils rencontrent le même problème que les sites de e-commerce : en raison de la grande quantité du contenu disponible, il est très difficile pour un utilisateur de trouver des articles qui répondent à ses besoins. Par conséquent, les réseaux P2P bénéficient beaucoup des systèmes de recommandation.

Malgré les nombreux avantages des systemes de recommandation décentralisés, la plupart des

travaux existants sur les algorithmes de recommandation a été concentré sur les systèmes centralisés à ce jour. Ces algorithmes ne sont donc pas directement applicables aux contextes distribués. Développer des systèmes de recommandation dans un contexte décentralisé est plus difficile que dans un contexte centralisé. A savoir, chaque utilisateur peut seulement tirer parti de sa propre information et des données fournies par un petit (par rapport à la taille du système) nombre des autres pairs[1]. Par conséquent, les pairs doivent avoir accès à un mécanisme décentralisé pour la détection efficace d'un nombre limité d'utilisateurs similaires parmi l'ensemble des pairs dans le système.

Nous comptons sur les algorithmes épidémiques comme la méthode de décentralisation pour détecter un ensemble d'utilisateurs similaires. Dans les protocoles épidémiques (également connu sous le nom *gossip protocols*), les pairs ont accès à un service d'échantillonnage aléatoire des pairs (Random Peer Sampling (RPS) [64]) leur fournissant constamment un sous-ensemble aléatoire des pairs du réseau. Quand un pair rejoint le réseau, sa vue est initialisée au hasard par le RPS. Chaque pair maintient également une *vue du réseau*. Cette vue contient des informations sur les c pairs qui maximisent une fonction de clustering. Dans notre travail, cette fonction de clustering reflète combien les pairs présentent un comportement similaire. Afin de converger vers la vue idéale, chaque pair gère un *protocole de regroupement (clustering protocol)* [127, 63]. Un pair sélectionne périodiquement une cible de gossip parmi les pairs de sa vue, et échange une partie de l'information de sa vue avec elle. Lors de la réception de nouvelles informations, le pair compare les nouveaux candidats avec sa vue actuelle, et un ensemble des pairs aléatoires proposés par le RPS. Puis, en gardant uniquement les c entrées les plus proches, il met à jour sa vue afin d'améliorer sa qualité. Bien que l'algorithme de clustering augmente les risques de partition du réseau, le RPS assure la connectivité avec une forte probabilité. Les protocoles de clustering épidémiques sont connus pour converger rapidement vers les vues de hautes qualités. En communiquant régulièrement avec les pairs de leur vue, ces protocoles aussi assurent leur vivacité et éliminent les noeuds déconnectés. Les protocoles épidémiques sont entièrement décentralisés, peuvent traiter un nombre elevé de départ de pair, et ne nécessitent pas de protocole spécifique pour être remis en marche après des échecs massifs.

Dans le Chapitre 2, tout d'abord les algorithmes de recommendation user-based et item-based sont mis en oeuvre dans un contexte P2P en utilisant des différentes mesures de similarité, et leur performance est comparée. Nous montrons que l'approche décentralisée user-based donne une bonne précision et moins de complexité que l'approache décentralisée item-based. En fait, les systèmes décentralisés du type user-based ne souffrent pas des inconvénients généralement attribués à leurs homologues centralisés, en particulier le manque de "scalability".

Deuxièmement, nous proposons un système de recommandation décentralisée basé sur les marches aléatoires. Le rôle des marches aléatoires est de réévaluer le poids de similarité entre les utilisateurs en fouillant des chemins plus longs dans le voisinage afin de trouver des similarités implicites transitives. Dans cet algorithme, chaque pair du système est pourvu d'un voisinage composé d'un petit (par rapport à la taille du système) nombre fixé d'autres pairs par le biais d'un protocole épidémique. Puis, les notes pour les articles inconnus dans le voisinage sont estimées en exécutant une marche aléatoire sur le voisinage. Une fois que les pairs ont formé leur voisinage, à savoir que le protocole épidémique a convergé, chaque pair est tout à fait indépendant des autres pairs pour générer ses recommandations. Cet algorithme fournit une meilleure performance que les algorithmes décentralisés plus simples du type user-based ou item-based, lorsque les données sont creuses (sparse).

[1] Les termes pair et utilisateur sont interchangeables dans ce document.

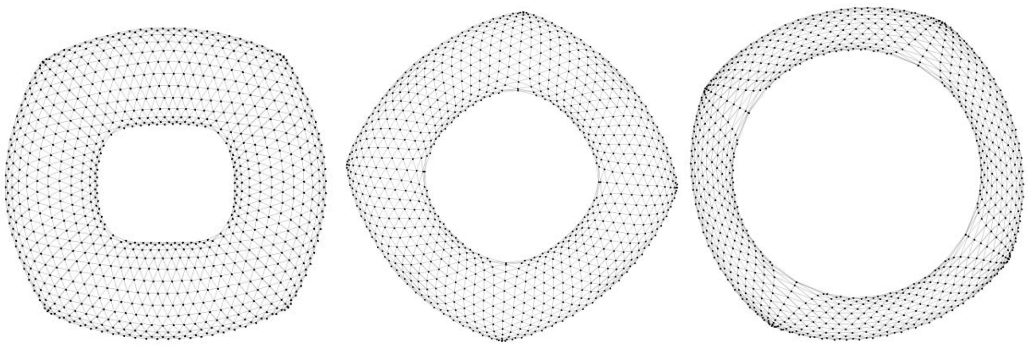
Troisièmement, le comportement de l'algorithme basée sur les marches aléatoires est discuté en détail en fonction de trois paramètres : le sparsity, la mesure de similarité et la taille du voisinage. Ce dernier devient particulièrement important dans le contexte P2P, car il affecte fortement la précision et la complexité de l'algorithme. Les valeurs des paramètres pour lesquels notre algorithme donne le meilleur rendement sont empiriquement trouvées pour l'ensemble de données MovieLens. L'algorithme de marche aléatoire améliore la précision sur un large éventail de sparsity tout en gardant le temps d'exécution à prix abordable pour les pairs.

5.5 Tracé de Graphes

Le tracé de graphes (graph drawing) est l'une des branches de visualisation de l'information avec des applications dans de nombreux domaines, y compris le militaire, la cartographie, les transports, la bioinformatique et l'analyse des réseaux sociaux. Le tracé d'un graphe appelé *layout* est la représentation graphique de ses sommets et de ses arêtes dans un espace Euclidien à 2 ou 3 dimensions. Nous nous concentrons sur le *straight-line graph drawing* où le problème de dessin est réduit à calculer les coordonnées des sommets. Une fois ces coordonnées trouvées, les arêtes sont représentées par les lignes droites entre les sommets correspondants. Deux approches importantes pour straight-line graph drawing sont les approches de force [40, 25, 68, 101, 36, 81] et les approches spectrales [74, 19, 80]. Les approches de force associent le layout avec une valeur scalaire appelée l'*énergie* du layout. Les coordonnées des sommets dans le layout optimal sont trouvées en minimisant la fonction d'énergie. Les approches spectrales définient une fonction de *l'énergie quadratique* pour le layout. Le problème de minimisation de cette fonction peut être reformulé comme un problème d'optimisation linéaire standard. Les solutions seront les vecteurs propres (généralisés) d'une matrice liée à la structure du graphe.

Les modèles de tracé de graphes révélent la structure du graphe sous-jacent en plaçant les sommets connectés l'un plus près de l'autre que les sommets non-connectés. En général, le niveau d'abstraction de la structure est fixé, c'est à dire, le rapport entre la longueur totale des arêtes et la distance totale entre tous les pairs de sommets est la même valeur pour toutes les exécutions d'un modèle sur le même graphe. Dans le chapitre 3, nous suggérons des approches de force ainsi que des approches spectrales pour le tracé de graphes en faisant varier le degré d'abstraction de la structure. Notamment, l'utilisateur peut contrôler par le biais d'un paramètre de *combien* les arêtes devraient être plus courts que les non-arêtes.

Nous proposons d'abord FlexGD, un algorithme flexible de force pour tracé de graphes. FlexGD dessine des graphes en fonction des deux critères de répartition uniforme des sommets et de l'abstraction de la structure. Le modèle est flexible, en ce qu'il est paramétré pour être biaisé vers l'un des deux critères du tracé, selon les préférences de l'utilisateur. En général, les équations de force ou d'énergie des modèles de force sont définies de telle sorte que les layouts résultants répondent à certains critères esthétiques de dessin comme la longueur uniforme des arêtes et la traversée minimale des arêtes. Pourtant, il a été démontré dans [101] que les critères dont on a parlé évitent les modèles classiques de la visualisation des clusters. Visualiser des clusters est important dans les applications telles que la détection de communauté dans les réseaux sociaux. Un nouveau modèle appelé *LinLog* est ensuite proposé par l'auteur qui est plus approprié pour atteindre à cet objectif. La fonction de l'énergie de FlexGD est étroitement liée à celle de LinLog, mais elle est subtilement modifiée afin d'améliorer ses caractéristiques. Plus précisément, nous remplaçons l'énergie de répulsion logarithmique du modèle LinLog par une énergie linéaire-logarithmique, tout en préser-



(a) $k = 600$ (b) $k = 300$ (c) $k = 1800$

FIGURE 5.3: Les layouts FlexGD du graphe jagmesh1 dans 3 niveaux d'abstraction.

vant l'attraction linéaire des arêtes intactes. Pour le layout $\mathbf{X}$ d'un graphe $G = (V, E)$, la fonction d'énergie de FlexGD est défini comme :

$$\text{FlexGD}(\mathbf{X}, k) \quad = \quad \sum_{\{i,j\} \in E} k\|x_i - x_j\| + \sum_{\{i,j\} \in V^{(2)}} \|x_i - x_j\| - \ln \|x_i - x_j\|.$$

Le premier terme capture la structure du graphe en raccourcissant les arêtes, tandis que le second terme distribue les sommets de façon uniforme sur la zone de dessin.

Les layouts de FlexGD ont les propriétés suivantes. La zone de dessin est rempli de manière optimale, et le layout est agréable dans les frontières. Deuxièmement, il améliore la propriété de la visualisation des clusters de LinLog à *visualisation réglable des clusters*, c'est-à-dire que l'utilisateur peut décider quelle densité les clusters doivent avoir et dans quelle mesure ils sont mis à part. Grace à cette propriété, la structure du graphe peut aussi être plus ou moins abstraite. Figure 5.3 montre les layouts du graphe jagmesh1 dans 3 niveaux d'abstraction. Enfin, le modèle est capable de dessiner les graphes déconnectés, à savoir que la plupart des modèles précédents ont des difficultés pour leur manipulation.

Nous dérivons formellement certaines propriétés de FlexGD, et les comparons avec celles de LinLog. La fonction d'énergie de FlexGD est minimisée par le biais d'une approche multi-niveaux. Dans les algorithmes multi-niveaux, le graphe est *durci* à une série de graphes plus petits. Chaque graphe durci doit refléter la structure du graphe plus fin. En outre, le calcul de son layout doit être beaucoup moins cher que le calcul du graphe plus fin. Le layout du graphe le plus durci est peu cher à calculer, car il est très petit en taille. Les coordonnées calculées des graphes durcis sont ensuite prolongés aux graphes plus fins. Le graphe plus fin a généralement besoin de moins de modifications, car il est déjà dans une forme approximativement bonne. Les algorithmes multi-niveaux conventionels fonctionnent bien si la distribution de la longueur d'arête est uniforme. Cependant, leur performance sur FlexGD n'est pas convaincante, parce que cette dernière crée une distribution non uniforme de la longueur de l'arête. L'algorithme multi-niveaux proposé pour FlexGD peut surmonter la distribution non-uniforme de la longueur d'arête. Les layouts FlexGD de certains grands ensembles de données réelles sont présentés pour illustrer le fait que l'algorithme peut générer des layouts de bonne qualité dans un large éventail d'abstractions, répondant aux préférences des différents utilisateurs.

Bien qu'aucun modèle unique ne puisse se vanter d'avoir de meilleures performances sur tous les graphes, l'aptitude d'un modèle de visualisation dépend de la topologie du graphe sous-jacent et des exigences de visualisation, il semble que dans de nombreux cas les layouts de FlexGD soient agréables autant que, parfois même plus que, les layouts des modèles précédents. En outre, l'utilisateur a plus de contrôle sur le layout avec FlexGD que les autres modèles grâce à son paramètre d'abstraction.

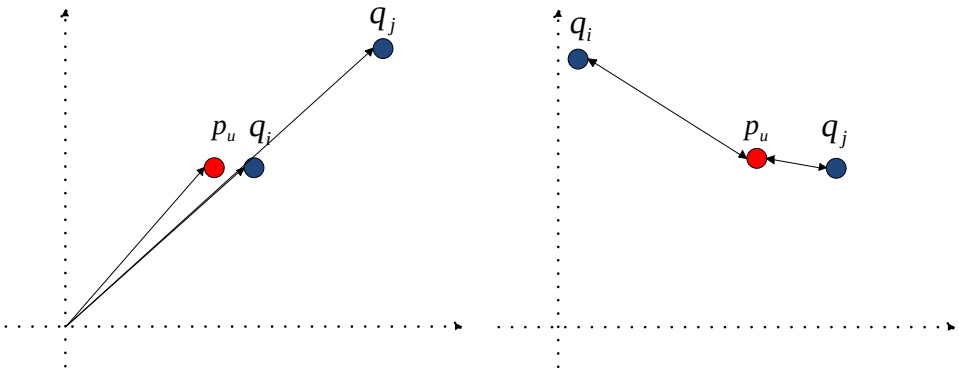
Nous montrons également comment l'approche spectrale peut être reformulée pour générer les layouts flexibles. Cette approche spectrale n'a pas les propriétés de clustering de FlexGD, mais son niveau d'abstraction est réglable. A la fin, la généralisation des approches de force et spectrale aux graphes *signés* est discutée. Dans les graphes signés les arêtes sont étiquetées comme positives ou négatives, là où les arêtes positives dénotent une force d'attraction et les arêtes négatives dénotent une force de répulsion entre les sommets connectés. Les graphes signés apparaissent dans les applications où les interactions au sein des données peuvent être négatives ou positives. Un exemple est les réseaux sociaux où les gens peuvent être amis ou ennemis.

5.6 Visualization des Donnees au Moyen du Filtrage Collaboratif

Suite à la competition Netflix, la plupart des travaux sur le FC ont été concentrés sur l'amélioration de la précision des prédictions [78, 90, 112]. Néanmoins, la précision n'est pas le seul facteur déterminant pour le succès d'un SR. Il est également important que les résultats soient représentés d'une façon agréable et informative. Le plus souvent, les recommandations sont présentées sous la forme d'une liste de classement. Les utilisateurs ont des intérêts multiples, et sont susceptibles de changer de préférences au fil du temps [130, 78, 73]. De la même manière, un compte d'utilisateur peut déléguer plusieurs personnes, chacune ayant un goût et des besoins différents. Neanmoins, les listes de classement ne peuvent pas dévoiler la corrélation entre les articles sauf si l'utilisateur les connait à priori. Ce defaut limite le bénéfice des systèmes de recommandation, car il empêche les utilisateurs d'effectuer une sélection correcte en fonction de leurs besoins actuels. Par exemple, un utilisateur de Netflix peut regarder des films documentaires au cours de la semaine, mais préfère les comédies d'action pour le week-end. Elle utilise également le même compte pour chercher des dessins animés pour ses enfants. En conséquence, sa liste de recommandation contient un mélange de différents genres. Cependant, il ne peut pas être compris à partir de la liste quel film est similaire au film d'action agréable qu'elle a vue pendant le week-end ou est plutôt comme le dessin animé que ses enfants ont apprécié la semaine dernière.

Dans le chapitre 4, nous suggérons une approache basée sur la factorisation de matrice (FM) pour visualiser les résultats du FC sous la forme de cartes bidimensionnelles. Cette dernière est une meilleure alternative aux listes de classement car elle organise les articles en fonction de leur similarité. Bien sûr, d'autres solutions, comme montrer de l'information sur le contenu des articles, peuvent également être envisagées pour améliorer l'informativité des recommandations. Cependant, cette solution nécessite des taches de collecte et de nettoyage compliquées que les entreprises essaient d'éviter. En outre, l'interprétation d'une carte visuelle est plus commode et assez intuitive. Comme le souligne l'adage "une image vaut mille mots".

Les systèmes de recommandations du type FM [34, 8, 122] projettent les articles et les utilisateurs dans un espace caractéristique latent. Les notes manquantes sont alors estimées par le produit intérieur des vecteurs de coordonnées de l'utilisateur et de l'article. La technique FM est également connue pour avoir la propriété de mettre les articles similaires proches les uns des autres dans l'es-



(a) Prédicteur du produit intérieur des vecteurs. j est plus pertinent à u que i, soit $p_u \cdot q_j > p_u \cdot q_i$, alors qu'il est plus loin d'elle, i.e. $\|p_u - q_j\| > \|p_u - q_i\|$.

(b) Prédicteur visuel. L'article plus pertinent j est plus proche de l'utilisateur u que l'article moins pertinent i. La distance entre l'origine et l'angle entre les vecteurs de coordonnées de l'utilisateur et des articles n'ont aucune importance.

FIGURE 5.4: La comparaison entre l'adéquation des fonction de la prédiction basées sur le produit intérieur et sur la distance Euclidienne pour la prédiction visuelle.

pace caractéristique latent [79]. Néanmoins, il y a deux obstacles à leur application aux cartes en 2 dimensions. Tout d'abord, les fonctions prédictives existantes ne sont pas adaptées à des fins de visualisation. Notamment, les utilisateurs relient intuitivement la pertinence entre les utilisateurs et les articles à leur distance Euclidienne sur la carte. Cependant, les fonctions prédictives existantes, généralement des variantes du produit intérieur des vecteurs, ont besoin d'autres informations (ex. l'angle entre le vecteur des coordonnees de l'utilisateur et le vecteur des coordonnées de l'article). Par conséquent, elle peuvent placer un article moins pertinent plus près d'un utilisateur qu'un article plus pertinent. Cette présentation est trompeuse, car les utilisateurs intuitivement interprètent les articles plus près comme plus pertinents. Plus précisément, le produit scalaire entre le vecteur des coordonnées d'un utilisateur, disons p_u, et le vecteur des coordonnées d'un article loins, disons q_j, peut-être une valeur plus grande que le produit intérieur de p_u et le vecteur des coordonnées d'un article à proximité, disons q_i. Un tel problème ne se pose pas, si les prévisions sont purement calculées en fonction de la distance Euclidienne, comme la pertinence entre un utilisateur et un article est calculé directement à partir de leur distance. La Figure 5.4 illustre pourquoi la fonction du produit intérieur n'est pas conforme aux exigences de visualisation, tandis que le prédicteur visuel basé sur la distance Euclidienne l'est. Deuxièmement, les approches basées sur la factorisation de matrice ont besoin de plus de 2 dimensions pour atteindre leur précision optimale. Par conséquent, les méthodes de projection adéquates préservant autant d'informations que possible sont requises.

Notre approche se compose de deux phases : la phase de intégration (embedding) et la phase de projection. La première phase intègre les utilisateurs et les articles dans un espace de grande dimension en utilisant un prédicteur visant à répondre aux besoins de visualisation. Ce prédicteur est une fonction décroissante de la distance Euclidienne entre l'utilisateur et l'article correspondant. Dans la deuxième phase, les données sont projetées à partir de l'espace de grande dimension dans une carte à 2 dimensions. Deux types de cartes sont présentés, chacun destiné à un but différent.

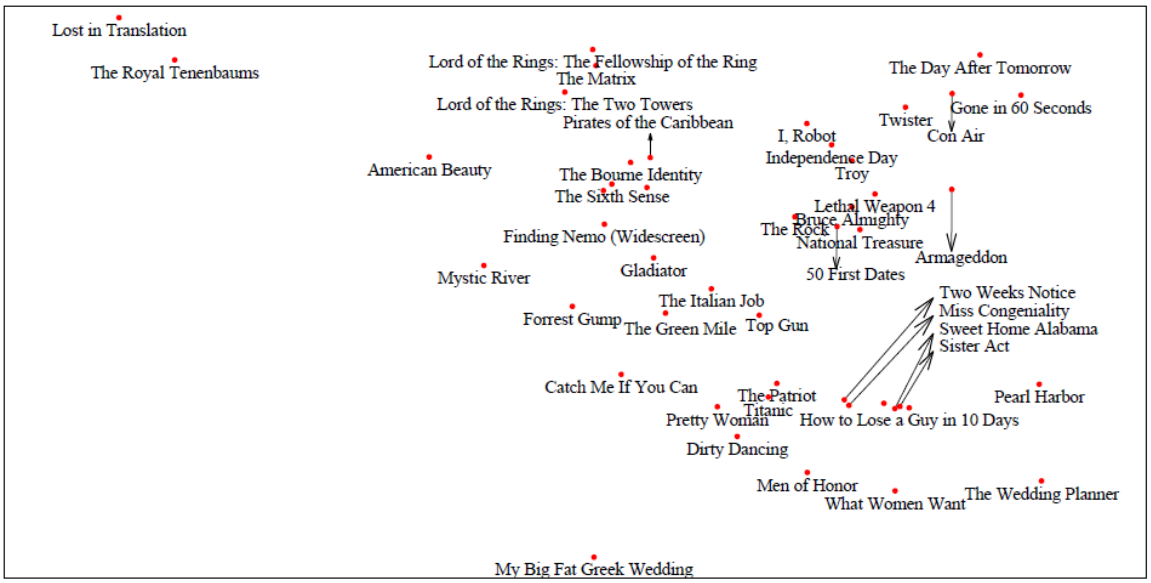
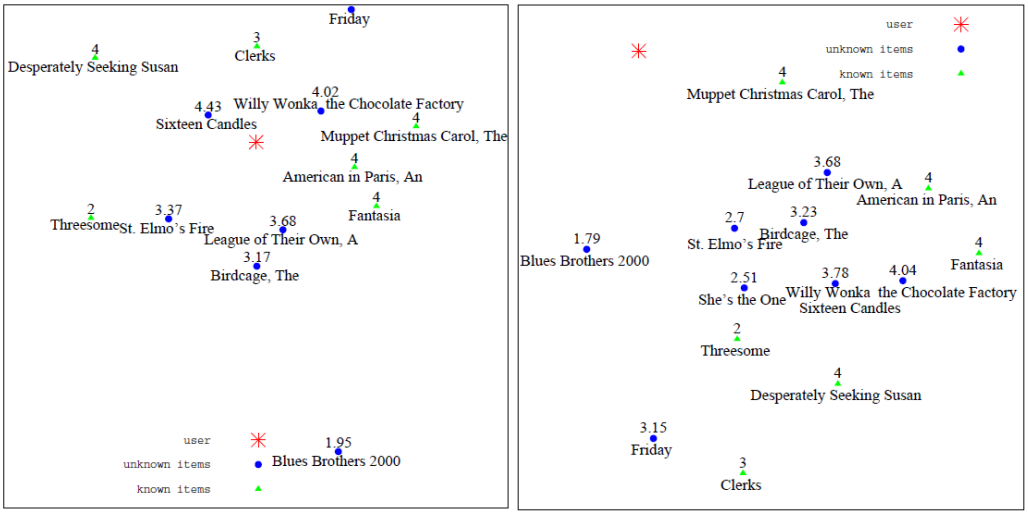


FIGURE 5.5: GIM des films Netflix projetées à partir d'un espace à 20 dimensions.

Une carte d'article globale (*Global Item Map (GIM)*) est générée à l'aide d'Analyse en Composantes Principales (ACP). GIM montre *l'ensemble des articles* dans l'espace bidimensionnelle en capturant le maximum de variance des données dans l'espace d'intégration originale. Elle est intéressante pour voir comment les articles sont regroupés en fonction de l'opinion générale des utilisateurs. Indépendamment de l'application sous-jacente, c'est une solution générale à la visualisation des données du FC. La Figure 5.5 montre comment les films populaires de l'ensemble de données Netflix Prize sont rangés sur la GIM. Avec peu de connaissances sur les films, nous pouvons voir que les films avec un contenu similaire sont proches les uns des autres tandis que ceux ayant un contenu différent sont loin les uns des autres.

Afin de représenter un *nombre limité d'articles* à un utilisateur spécifique, nous suggérons les Cartes d'article personnalisées (*Personalized Item Maps(PIMS)*). Elles sont intéressantes dans les applications comme les recommandations visuelles. Une PIM met les articles autour de l'utilisateur actif de telle sorte que leur pertinence diminue avec leur distance. La PIM préfère préserver la *structure locale* de l'espace de grande dimension. En d'autres termes, sa priorité est de préserver les distances courtes plutôt que les distances longues. Les PIMs sont générés en utilisant l'analyse en composantes curvilineaires (Curvilinear Component Analysis (CCA)). L'approche originale du CCA est modifiée afin de donner plus d'importance à la préservation des distances par rapport à l'utilisateur actif. Toutes les PIMs sont présentés avec les mêmes paramètres visuels afin que la comparaison entre les PIMs des différents utilisateurs soit significative. La Figure 5.6 compare les PIMs d'un utilisateur de MovieLens projeté de l'espace caractéristique latente visuel et l'espace caractéristique latente du produit intérieur des vecteurs. Nous projetons à partir d'un espace caractéristique latent visuel à 20 dimensions un espace caractéristique latent du produit intérieur à 10 dimensions, parce que 20 et 10 dimensions correspondent à la meilleure précision de chaque algorithme, respectivement. Comme le montrent les figures, l'utilisateur est entouré de quelques films inconnus pour lui et d'un certain nombre de films déjà notés par elle. Les notes des films connus sont



(a) La PIM du prédictor visuel projetée d'un espace caractéristique latente à 20 dimensions.

(b) La PIM du produit intérieur des vecteurs projetée d'un espace caractéristique latente à 10 dimensions.

FIGURE 5.6: La PIM d'un utilisateur anonyme MovieLens.

les notes réelles de l'utilisateur. Les notes des items inconnus sont les notes prédites, calculées dans l'espace caractéristique latent de grande dimension. Remarquez que la note prévue est une fonction décroissante de la distance dans l'espace caractéristique latent visuel. Par conséquent, la cohérence entre les notes prédites et la distance entre l'utilisateur et l'article correspondant dans la PIM du prédicteur visuel est une mesure pour observer combien les distances originales ont été conservées. On voit que dans la PIM générée à partir de *l'espace caractéristique latente visuelle* (figure 5.6a) que les articles plus pertinents sont plus proches de l'utilisateur que les articles moins pertinents.

La précision de notre approche est validée en menant des expériences sur les trois versions de l'ensemble de données MovieLens et l'ensemble de données Netflix. La concentration dans ce chapitre porte sur la recommandation pour les films. Néanmoins, la même approche peut être utilisée pour tout autre type de données comme les musiques, les restaurants, les livres, etc.

La élégance de cette approche est sa capacité à communiquer des informations latentes qui sont difficiles à comprendre, même si on dispose de l'information sur le contenu. En effet, les cartes présentées sont le résultat de la sagesse collective d'un grand nombre d'utilisateurs, étant parfois plus prometteur que chercher dans des tonnes d'information générée par les experts. L'application des techniques proposées ne se limite pas aux recommandations. Par exemple GIM peut être utilisé pour surveiller l'opinion publique sur les produits. De même, les PIMs peuvent être utilisées pour représenter une comparaison visuelle entre les éléments d'un *catalogue numérique*.

Une autre application potentielle des méthodes suggérées est de fonder une base pour mesurer la diversité d'une liste de recommandations. Rappelez-vous que la similarité entre les articles est une fonction de leur distance Euclidienne sur la GIP. En outre, les axes ne sont pas corrélés et chacun contient de nouvelles informations sur les différents caractéristiques des articles. Par conséquent, la variance des coordonnées des articles d'une liste de recommandation montre comment la diversité des recommandations. Si la liste de recommandation d'un utilisateur u est représentée par le $\hat{R}_u$, la

diversité de cette liste est définie comme suit :

$$\mathrm{var}(\hat{R}_u) = \sqrt{\frac{\sum_{\hat{r}_{ui} \in \hat{R}_u} (\|q_i - \bar{q}(u)\|)^2}{\left|\hat{R}_u\right|}},$$

où $\bar{q}(u)$ est le centre des coordonnées des articles dans la liste des recommandations pour u.

Il convient également de mentionner que, bien que nous ayons discuté seulement des cartes d'articles, la même approche peut être appliquée pour les utilisateurs. Par exemple, une carte d'utilisateurs globale peut être générée pour détecter les groupes sociaux de consommateurs montrant les mêmes habitudes dans leur comportement de consommation.

BIBLIOGRAPHY

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

[2] Baruch Awerbuch, Yossi Azar, Zvi Lotker, and Boaz Patt-Shamir. Collaborate with strangers to find own preferences. *Theory of Computing Systems*, 42(1), 2008.

[3] Yossi Azar, Amos Fiat, Anna R. Karlin, Frank Mcsherry, and Jared Saia. Spectral analysis of data. In *Proc. of the thirty-third annual ACM symposium on Theory of computing*, pages 619 – 626, 2001.

[4] Marko Balabanovic and Yoav Shoham. Fab: content-based, collaborative recommendation. In *Communications of the ACM*, pages 66–72, 1997.

[5] Josh Barnes and Piet Hut. A hierarchical $o(nlogn)$ force-calculation algorithm. In *Nature*, pages 446–449, 1986.

[6] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 714–720. AAAI Press, 1998.

[7] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.

[8] Robert Bell, Yehuda Koren, and Chris Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104, 2007.

[9] Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, pages 75–79, 2007.

[10] James Bennett and Stan Lanning. The netflix prize. In *KDD Cup and Workshop*, 2007.

[11] Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, pages 41–88, 1999.

[12] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, pages 993–1022, 2003.

[13] I. Borg and P. J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications (Springer Series in Statistics)*. Springer, 2005.

[14] Derek Bridge, Mehmet H. Göker, Lorraine McGinty, and Barry Smyth. Case-based recommender systems. *Knowl. Eng. Rev.*, pages 315–320, 2005.

[15] Michael J. Brzozowski, Tad Hogg, and Gabor Szabo. Friends and foes: ideological social networking. In *SIGCHI conference on Human factors in computing systems*, pages 817–820, 2008.

[16] Robin Burke. The wasabi personal shopper: A case-based recommender system. In *IN PROCEEDINGS OF THE 11TH NATIONAL CONFERENCE ON INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE*, pages 844–849. AAAI Press, 1999.

[17] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, pages 331–370, 2002.

[18] John Canny and Stephen Sorkin. Practical large-scale distributed key generation. In *Advances in Cryptology*, pages 138–152, 2004.

[19] Ali Çivril, Malik Magdon-Ismail, and Eli Bocek-Rivele. Ssde: fast graph drawing using sampled spectral distance embedding. In *Proceedings of the 14th international conference on Graph drawing*, pages 30–41, 2006.

[20] V.Y. Chen, C.Z. Qian, and R.F. Woodbury. Visualizing collaborative filtering in digital collections. In *Information Visualization, 2007. IV '07. 11th International Conference*, pages 203 –210, 2007.

[21] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[22] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *In Proceedings of ACM SIGIR Workshop on Recommender Systems*, 1999.

[23] Trevor F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2000.

[24] Sanjoy Dasgupta, Wee Sun Lee, and Philip M. Long. A theoretical analysis of query selection for collaborative filtering. In *Machine Learning*, page 2003, 2003.

[25] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. In *ACM Transactions on Graphics*, pages 301–331, 1996.

[26] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, pages 1:1–1:25, 2011.

[27] Marco de Gemmis, Pasquale Lops, Giovanni Semeraro, and Pierpaolo Basile. Integrating tags in a semantic content-based recommender. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 163–170. ACM, 2008.

[28] Jan de Leeuw. Applications of convex analysis to multidimensional scaling. In *Recent Developments in Statistics*, pages 133–146. North Holland Publishing Company, 1977.

[29] Toon De Pessemier, Tom Deryckere, and Luc Martens. Context aware recommendations for user-generated content on a social network site. In *Proceedings of the seventh european conference on European interactive television conference*, pages 133–136, 2009.

[30] P. Demartines and J. Herault. Curvilinear component analysis: a self-organizing neural network for nonlinear mapping of data sets. *Neural Networks, IEEE Transactions on*, pages 148 –154, 1997.

[31] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, pages 1–12, 1987.

[32] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.

[33] Petros Drineas, Iordanis Kerenidis, and Prabhakar Raghavan. Competitive recommendation systems. In *Proc. of the thiry-fourth annual ACM symposium on Theory of computing*, pages 82–90, 2002.

[34] Petros Drineas, Iordanis Kerenidis, and Prabhakar Raghavan. Competitive recommendation systems. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 82–90, New York, NY, USA, 2002. ACM.

[35] I.S. Duff. A survey of sparse matrix research. *Proceedings of the IEEE*, pages 500 – 535, 1977.

[36] Peter Eades. A heuristic for graph drawing. In *Congressus Numerantium*, pages 149–160, 1984.

[37] A. Felfernig and R. Burke. Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th international conference on Electronic commerce*, pages 3:1–3:10. ACM, 2008.

[38] Karlis Freivalds, Ugur Dogrusoz, and Paulis Kikusts. Disconnected graph layout and the polyomino packing approach. In *Graph Drawing*, pages 572–575. Springer Berlin / Heidelberg, 2002.

[39] Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings of the DIMACS International Workshop on Graph Drawing*, GD '94, pages 388–403, 1995.

[40] Thomas M.J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. In *Software-Practice and Experience*, pages 1129–1164, 1991.

[41] Emden R. Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *GRAPH DRAWING*, pages 239–250. Springer, 2004.

[42] Jennifer Golbeck. Generating predictive movie recommendations from trust in social networks. In *Proceedings of the 4th international conference on Trust Management*, pages 93–104. Springer-Verlag, 2006.

[43] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, 1996.

[44] Google code. *matrix-toolkits-java*, 2012. `http://code.google.com/p/matrix-toolkits-java/`.

[45] Georg Groh. Recommendations in taste related domains: Collaborative filtering vs. social filtering. In *In Proc ACM GroupŠ07*, pages 127–136, 2007.

[46] GroupLens. *MovieLens Datasets*, 2010. `http://www.grouplens.org/node/73#attachments`.

[47] Anshul Gupta, George Karypis, and Vipin Kumar. Highly scalable parallel algorithms for sparse matrix factorization. Technical report, IEEE Transactions on Parallel and Distributed Systems, 1994.

[48] Stefan Hachul and Michael Jünger. Large-graph layout algorithms at work: An experimental study. *Journal of Graph Algorithms and Applications*, pages 345–369, 2007.

[49] P. Hage and F. Harary. *Structural Models in Anthro-pology*. Cambridge University Press, 1983.

[50] Kenneth M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, pages 219–229, 1970.

[51] David Harel and Yehuda Koren. A fast multi-scale method for drawing large graphs. In *Journal of Graph Algorithms and Applications*, 2002.

[52] David Harel and Yehuda Koren. Graph drawing by high-dimensional embedding. In *Revised Papers from the 10th International Symp. on Graph Drawing*, pages 207–219, 2002.

[53] Fritz Heider. Attitudes and cognitive organization. *The Journal of Psychology*, pages 107–112, 1946.

[54] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. of the 22nd annual Int. ACM SIGIR*, pages 230–237, 1999.

[55] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, pages 5–53, 2004.

[56] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, pages 89–115, 2004.

[57] Y. F. Hu. Efficient and high quality force-directed graph drawing. *The Mathematica Journal*, pages 37–71, 2005.

[58] Y. F. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM J. Sci. Comput.*, pages 1352–1375, 2001.

[59] Zan Huang, Wingyan Chung, Thian-Huat Ong, and Hsinchun Chen. A graph-based recommender system for digital library. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 65–73, 2002.

[60] Mohsen Jamali and Martin Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proc. of the 15th ACM SIGKDD*, pages 397–406, 2009.

[61] Bennett James and Stan Lanning. *The Netflix Prize*, 2007. KDD Cup and Workshops.

[62] D. Jannach. Finding preferred query relaxations in content-based recommenders. In *Intelligent Systems, 2006 3rd International IEEE Conference on*, pages 355 –360, 2006.

[63] Mark Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-man: Gossip-based fast overlay topology construction. *IJCNC*, 2009.

[64] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.

[65] Xin Jin, Yanzan Zhou, and Bamshad Mobasher. A maximum entropy web recommendation system: combining collaborative and content features. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 612–617, 2005.

[66] Michael J.Pazzani. A framework for collaborative, content-based and demographic filtering. In *Artificial Intelligence Review*, pages 393–408, 1999.

[67] Jung Project. *Java Universal Network/Graph Framework*, 2012. `http://jung.sourceforge.net`.

[68] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, pages 7–15, 1989.

[69] J. Kamahara, T. Asakawa, S. Shimojo, and H. Miyahara. A community-based recommendation system to reveal unexpected interests. In *Multimedia Modelling Conference, 2005. MMM 2005. Proceedings of the 11th International*, pages 433 – 438, 2005.

[70] Michael Kaufmann and Dorothea Wagner, editors. *Drawing graphs: methods and models*. Springer-Verlag, 2001.

[71] Anne-Marie Kermarrec and Maarten Van Steen. Gossiping in destributed systems. *ACM SIGOPS Operating Systems Review*, pages 2–7, 2007.

[72] Mohammad Khoshneshin and W. Nick Street. Collaborative filtering via euclidean embedding. In *ACM conference on Recommender systems*, RecSys '10, 2010.

[73] J. Zico Kolter and Marcus A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res.*, pages 2755–2790, 2007.

[74] Y. Koren. Drawing graphs by eigenvectors: theory and practice. *Comput. Math. Appl.*, pages 1867–1888, 2005.

[75] Y. Koren. Drawing graphs by eigenvectors: theory and practice. *Comput. Math. Appl.*, pages 1867–1888, 2005.

[76] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of the 14th ACM SIGKDD*, pages 426–434, 2008.

[77] Yehuda Koren. The bellkor solution to the netflix grand prize. Technical report, Netflix Prize Report, 2009.

[78] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proc. of the 15th ACM SIGKDD*, pages 447–456, 2009.

[79] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, pages 30–37, 2009.

[80] Yehuda Koren, L. Carmel, and D. Harel. Ace: a fast multiscale eigenvectors computation for drawing huge graphs. In *INFOVIS*, pages 137–144, 2002.

[81] Yehuda Koren and Ali Çivril. The binary stress model for graph drawing. In *Graph Drawing*, pages 193–205. Springer-Verlag, 2009.

[82] Bruce Krulwich. Lifestyle finder: Intelligent user profiling using large-scale demographic data. *AI Magazine*, pages 37–45, 1997.

[83] Michael Kuhn, Roger Wattenhofer, and Samuel Welten. Social audio features for advanced music retrieval interfaces. In *international conference on Multimedia*, pages 411–420, 2010.

[84] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Recommendation systems: A probabilistic analysis. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 664–673, 1998.

[85] Jérôme Kunegis, Andreas Lommatzsch, and Christian Bauckhage. The slashdot zoo: mining a social network with negative edges. In *World wide web (WWW)*, pages 741–750, 2009.

[86] Jérôme Kunegis, Stephan Schmidt, Andreas Lommatzsch, Jürgen Lerner, Ernesto William De Luca, and Sahin Albayrak. Spectral analysis of signed graphs for clustering, prediction and visualization. In *SDM*, pages 559–570, 2010.

[87] Neal Lathia. *Evaluating Collaborative Filtering Over Time*. PhD thesis, University College London, 2010.

[88] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. In *Internet Computing, IEEE*, pages 76–80, 2003.

[89] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, pages 395–416, 2007.

[90] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *IEEE International Conference on Data Mining*, pages 43–52, 2007.

[91] Hao Ma, Haixuan Yang, Michael R. Lyu, and Irwin King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 931–940. ACM, 2008.

[92] Tariq Mahmood and Francesco Ricci. Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, HT '09, pages 73–82. ACM, 2009.

[93] Frank McSherry and Ilya Mironov. Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 627–636. ACM, 2009.

[94] Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth national conference on Artificial intelligence*, pages 187–192, 2002.

[95] Rada Mihalcea. Graph-based ranking algorithms for sentence extraction, applied to text summarization. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, 2004.

[96] Bradley N. Miller, Joseph A. Konstan, and John Riedl. Pocketlens: Toward a personal recommender system. *ACM Trans. Inf. Syst.*, 22(3):437–476, 2004.

[97] Raymond J. Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[98] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, 2006.

[99] Netflix. *Netflix Prize*, 2010. `www.netflixprize.com/`.

[100] M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, pages 167–256, 2003.

[101] Andreas Noack. Energy models for graph clustering. *Journal of Graph Algorithms and Applications*, pages 453–480, 2007.

[102] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[103] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07*, pages 39–42, 2007.

[104] D. Pavlov, E. Manavoglu, C.L. Giles, and D.M. Pennock. Collaborative filtering with maximum entropy. *Intelligent Systems, IEEE*, pages 40 – 47, 2004.

[105] Michael Pazzani and Daniel Billsus. Content-based recommendation systems. In *The Adaptive Web*, pages 325–341. Springer Berlin / Heidelberg, 2007.

[106] Martin Piotte and M. Chabbert. The pragmatic theory solution to the netflix grand prize. Technical report, Netflix Prize Report, 2009.

[107] Alexandrin Popescul, David M. Pennock, and Steve Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 437–444. Morgan Kaufmann Publishers Inc., 2001.

[108] Thomas Puppe. *Spectral Graph Drawing: A Survey*. VDM Verlag, 2008.

[109] Aaron Quigley and Peter Eades. Fade: Graph drawing, clustering, and visual abstraction. In *Graph Drawing*, pages 197–210, 2001.

[110] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, and John Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM, 2002.

[111] John S. Breeze, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, 1998.

[112] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, 2007.

[113] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proc. of the 24th international conference on Machine learning*, pages 791–798, 2007.

[114] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Trans. Comput.*, pages 401–409, 1969.

[115] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proc. of the 10th Int. Conf. on World Wide Web*, pages 285–295, 2001.

[116] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, pages 27 – 64, 2007.

[117] Andrew I. Schein, Alexandrin Popescul, Lyle H., Rin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 253–260. ACM Press, 2002.

[118] Mark Van Setten, Stanislav Pokraev, Johan Koolwaaij, and Telematica Instituut. Context-aware recommendations in the mobile tourist application compass. In *In Nejdl, W. and De Bra, P. (Eds.). AH 2004, LNCS 3137*, pages 235–244, 2004.

[119] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995.

[120] Barry Smyth. The adaptive web. In *The Adaptive Web*, chapter Case-based recommendation, pages 342–376. Springer-Verlag, 2007.

[121] Masashi Sugiyama. Active learning in approximately linear regression based on conditional expectation of generalization error. *JOURNAL OF MACHINE LEARNING RESEARCH*, 7:141–166, 2006.

[122] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, pages 623–656, 2009.

[123] Tribler. *Tribler*, 2010. `http://www.tribler.org`.

[124] Andreas Töscher, Michael Jahrer, and Robert M. Bell. The bigchaos solution to the netflix grand prize. Technical report, Netflix Prize Report, 2009.

[125] Daniel Tunkelang. Jiggle: Java interactive graph layout environment. In *Graph Drawing (GD)*, pages 413–422, 1998.

[126] Spyros Voulgaris and Maarten Van Steen. Epidemic-style management of semantic overlays for content-based searching. In *In EuroPar*, pages 1143–1152. Springer, 2005.

[127] Spyros Voulgaris and Maarten Van Steen. Epidemic-style management of semantic overlays for content-based searching. In *EuroPar*, pages 1143–1152, 2005.

[128] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Graph Drawing*, pages 31–55, 2001.

[129] C. Walshaw, M. G. Everett, and M. Cross. Parallel dynamic graph partitioning for adaptive unstructured meshes. *J. Parallel Distrib. Comput.*, pages 102–108, 1997.

[130] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.*, pages 69–101, 1996.

[131] W. Woerndl, C. Schueller, and R. Wojtech. A hybrid recommender system for context-aware recommendations of mobile applications. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, 2007.

[132] Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD*, pages 723–732, 2010.

[133] Rebecca Xiong, Marc A. Smith, and Steven M. Drucker. Visualizations of collaborative information for end-users. Technical report, Microsoft, 1999.

[134] Hilmi Yildirim and Mukkai S.Krishnamoorthy. A random walk method for alleviating the sparsity problem in collaborative filtering. In *Proc. of the ACM Conf. on Recommender systems*, pages 131–138, 2008.

[135] Zhiwen Yu, Xingshe Zhou, Daqing Zhang, Chung-Yau Chin, Xiaohang Wang, and Ji men. Supporting context-aware media recommendations for smart phones. *Pervasive Computing, IEEE*, pages 68 –75, 2006.

[136] Markus Zanker. A collaborative constraint-based meta-level recommender. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 139–146. ACM, 2008.

[137] C. Lawrence Zitnick and Takeo Kanade. Maximum entropy for collaborative filtering. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 636–643, 2004.