Technical University of Munich

Department of Informatics

Bachelor's Thesis in Informatics

Network Learning via Ranking

Thomas Edlich

Technical University
of Munich

Department of Informatics

Bachelor's Thesis in Informatics

Network Learning via Ranking

Netzwerk Lernen via Ranking

Author:	Thomas Edlich
Supervisor:	Prof. Dr. Stephan Günnemann
Advisors:	Oleksandr Shchur, M.Sc.
	Aleksandar Bojchevski, M.Sc.
Submission:	21.08.2017

I assure that this bachelor's thesis is my own work and I have documented all sources and material used.

München, 21.08.2017

....
(Thomas Edlich)

Acknowledgements

I would like to thank Prof. Dr. Stephan Günnemann for giving me the opportunity to write this thesis at his chair.

Furthermore, I would like to acknowledge my supervisors Oleksandr Shchur and Aleksandar Bojchevski for their amazing guidance during this semester. This thesis would not have been possible without their critical questions and their helpful advices.

Lastly, I want to express my gratitude for friends and family, who encouraged me throughout this thesis, spent many hours in the library with me and read and criticised this thesis.

Abstract

Networks are used in a variety of scientific fields to model arbitrarily large and complex systems. Therefore many graph algorithms have been developed to solve tasks like clustering or link prediction. The outcome of many of these algorithms is influenced by edge weights. Unfortunately, these weights might often not be given which could lead to suboptimal performance. In many cases there is, however, some knowledge about a ranking which describes the importance of a certain node in the network.

This thesis aims to develop methods to recover edge weights for these cases. Essentially, we intend to solve an inverse ranking problem where an unweighted graph and a ranking are given and the goal is to find an edge weight assignment such that the ranking computed on the weighted graph corresponds to the given target ranking. We introduce algorithms which are able to solve this weight recovery task if they are either given the numerical ranking scores for nodes or even if only a ranking order is given. These algorithms also work if only a partial ranking consisting of the scores or the order of the k most important nodes is given.

We furthermore evaluate whether Network Learning can have a positive impact on Data Mining and Machine Learning algorithms on graphs by conducting experiments on the influence of edge weights on the performance of spectral clustering and link prediction.

Contents

Symbols and Notation

Symbol	Meaning
$\mathbb{1}$	Indicator function: $\mathbb{1}(cond) = \begin{cases} 1, \text{if } cond = \text{true} \\ 0, \text{else} \end{cases}$
I	The identity matrix.
$[n]$	The set of natural numbers up to n such that $[n] = \{1, 2, \ldots, n\}$
c	The portability factor for PageRank. In this thesis we always assume $c = 0.85$.
q	A stochastic query vector used for personalized PageRank. $q \in [0,1]^n, \|q\|_1 = 1$
Q	The set of query vectors.
r	A computed PageRank vector. $r \in [0,1]^n, \|r\|_1 = 1$. r_q denotes the personalized PageRank vector for a query vector q.
$\overline{r}$	A target PageRank vector. $r \in [0,1]^n, \|r\|_1 = 1$. $\overline{r}_q$ denotes the personalized PageRank vector for a query vector q.
R	A computed PageRank order vector. R is a permutation of $[n]$ and R_i corresponds to the position of node v_i in a sorted vector of PageRank scores.
$\overline{R}$	A target PageRank order vector. $\overline{R}$ is a permutation of $[n]$ and $\overline{R}_i$ corresponds to the position of node v_i in a sorted vector of PageRank scores.
G_u	An unweighted graph $G_u = (V, E)$.
G_w	A weighted graph $G_u = (V, E, W)$.
V	The set of nodes of a graph G.
E	The set of edges of a graph G.
n	The amount of vertices in a network such that $n = \|V\|$.
m	The amount of edges in a network such that $m = \|E\|$.
w	A vector of weights such that $w \in \mathbb{R}^m$.
A	The adjacency matrix of a graph G. $A \in \mathbb{R}^{n \times n}$.
k	The number of known top scores in the top-k scenario.

Part I

Introduction

Chapter 1

Introduction

Networks are a popular way to represent interactions in arbitrarily large and complex systems and are therefore used in many scientific areas. In biology a graph might for example represent a protein-protein interaction network [1], while social scientists could use graphs to model dependencies between historical events [2]. Other applications of graphs include power-grids, the web and supply-chain-management [3, 4].

Due to their popularity, there has been much effort to develop algorithms to extract information from a network. These graph mining methods span a variety of tasks including finding optimal paths from a source to a target, detecting communities of associated nodes and the prediction or recommendation of new connections. Most of these algorithms use the graph structure, i.e. which node is connected to which other nodes, however better performance can often be achieved by also considering additional information about nodes or edges called node and edge features. Node features might for example include personal data like name or age in a social network or maybe paper title and author in a citation network while edge features might be weights or time stamps. Edge weights are a commonly used feature since they make it possible to quantify the connection between nodes. This measure might for example represent a distance, a similarity or quantification of the strength of the connection. Having these weights is often more beneficial than only knowing of the existence of an edge. Consider for example a road network where edge weights represent the length of a road. This additional information makes it possible to find a shortest path regarding the actual driving distance instead of the number of roads.

Unfortunately, in many cases the graph structure might be known but not the actual edge weights which could diminish the performance of graph algorithms. However, it could be possible that a ranking, i.e. a measure of importance, is known for all or some nodes in the network. Consider for example the case of a social network where one would like to find social circles based on the strength of connections between users and assume that the connection structure of the social network is completely known. One might then know for some users which other users they interact with most. This interaction frequency might either be given as a numerical value, e.g. 50% of user A's communication is with user B, 25% with user C and so on. Or only the order of the communication frequency might be known, i.e. user A communicates the most with user B and second most with user C. Based on this ranking one would then like to find weights for each connection in the network in order to find the social circles.

1.1 Objectives

This thesis aims to develop methods to recover edge weights for a given network where additionally some kind of ranking is known. Furthermore, we intend to determine if it is necessary that a ranking score for each node in the network is given or if a partial ranking where only some scores are known is sufficient to recover edge weights. Moreover, we aim to find a method that does not require numerical ranking scores but only a ranking order to find a suitable edge weight assignment. This ranking order might again be complete such that the ranking position of every node is known or only partial such that only the ranking positions of the most important nodes are given.

1.2 Outline

This thesis is structured as follows. After this introduction, we continue by presenting a brief background review regarding ranking algorithms, spectral clustering and link prediction in Chapter 2. There we furthermore discuss some related work and explain what discriminates our approach from previous methods.

Part II then contains our developed methods for learning edge weights for a given graph and given rankings. Chapter 3 presents our first approach in which we learn weights by minimizing a loss function between the given ranking scores and computed scores. In Chapter 4 we then extend this method to cases where only the ranking order is known but not the actual ranking scores.

The final part of this thesis is about analysing the effectiveness of our approach. Chapter 5 briefly discuss the setup of our conducted experiments, including details about the datasets we used and our implementation. Chapter 6 and 7 then present numerical results of our methods. Furthermore, we show how our approach impacts the performance of spectral clustering and link prediction in Chapter 8 and 9. Finally, we summarize our work, draw conclusions and discuss future work regarding Network Learning in Chapter 10.

Chapter 2

Background and Related Work

In this chapter we explain necessary background knowledge and summarize related work. First, an introduction to ranking in networks and especially the ranking algorithm PageRank is presented. Afterwards we explain the fundamentals of spectral clustering and link prediction. Finally we give an overview of related projects.

2.1 Ranking in Networks

Ranking algorithms for networks use certain features of a graph in order to calculate the importance of every node. One of the most used features is the network structure, i.e. the edges and possibly further edge information like weights. More formally, let an unweighted network G_u be defined by $G_u = (V, E)$ where V is a set of nodes and $E \subseteq V \times V$ is the set of edges in the graph. A weighted graph G_w is then obtained by adding a set of weights W, such that $G_w = (V, E, W)$. A ranking algorithm r can then be defined as

$$r : V \to \mathbb{R}. \tag{2.1}$$

Essentially, Equation (2.1) assigns each node $v \in V$ a global ranking score, measuring the importance of v in network G.

2.1.1 PageRank

PageRank was introduced in 1998 by Larry Page et al. [5] in order to measure the importance of web pages based on the graph representation of the web. Since then it has become a well-known unsupervised solution for ranking problems [6]. The main idea of the PageRank algorithm is that a web page is important if many other important web pages contain links to it. A simplified version of PageRank is then defined as [7]:

$$PR(u) = \alpha \sum_{v \in B(u)} \frac{PR(v)}{N_v}, \tag{2.2}$$

where $B(u)$ denotes the set of all web pages that point to u, N_v the number of outgoing links from v, $PR(u)$ and $PR(v)$ the PageRank scores of node u and v and α a normalization factor. The PageRank scores for all nodes can then be iteratively calculated.

Unfortunately there is a problem with this simplified PageRank version. Consider two nodes which only have outgoing links to each other but have incoming links from other nodes. This loop would then accumulate ranking scores from other pages but never distribute it back. Page et al. called this a *rank sink* [5].

PageRank essentially models a random walk on the web graph where each outgoing link of a node is picked with the same probability. The PageRank scores then correspond to the standing probability distribution of this random walk [5]. This model can be modified in order to deal with the above mentioned rank sink issue by allowing the random walk to jump to a random node in the network with probability $(1 - c)$ instead of choosing an edge. The updated PageRank model is then defined as [7]:

$$PR(u) = (1 - c) + c \sum_{v \in B(u)} \frac{PR(v)}{N_v}. \tag{2.3}$$

For the remainder of this thesis we will assume $c = 0.85$. Equation (2.3) can also be reformulated to matrix notation [5]:

$$r = cAr + (1 - c)q, \tag{2.4}$$

or in terms of r:

$$r = (1 - c)(I - cA)^{-1}q, \tag{2.5}$$

where $r \in \mathbb{R}^n$ represents the ranking vector such that $r_i = PR(v_i)$. A denotes the adjacency matrix of the web graph and I the identity matrix. We call the stochastic vector $q \in \mathbb{R}^n$ a *query vector*, which determines the probability distribution where the random walk jumps to.

If the query vector q is set to a uniformly distributed stochastic vector, i.e. $q = \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \dots & \frac{1}{n} \end{pmatrix}$, the PageRank vector r corresponds to a *global* ranking. Page et al. [5] however also discussed how q can be leveraged to achieve a *personalized* ranking. As mentioned before, q models the distribution where the random walk jumps to instead of following a link. Intuitively, this describes the probability that a user enters a new URL in a browser instead of following a link on his current website. A uniform query vector in this context would essentially mean that a user is equally likely to enter any URL. However, this does not actually correspond to real user behaviour since most users have preferred websites they use very regularly while they almost never visit other websites. Therefore, one can modify q in order to model the real behaviour of the user by increasing q_i if a user often visits web page v_i or decreasing it if the user rarely visits v_i. The real distribution could for example be learned by analysing the web page history of a user.

In other cases it might make sense to set q to a unit vector such that it consists entirely of zeros while only a single entry q_i is set to 1 such that the random walk either follows a link in the network or jumps back to q_i. In consequence, nodes farther away from q_i receive a lower ranking score since the probability of reaching them is rather low while nodes close to q_i receive higher rankings.

The original PageRank algorithm of Page et al. [5] only considered uniform edge weights such that all outgoing edges had a weight of $\frac{1}{N_v}$, however it is easy to modify Equation (2.3) such that it supports arbitrary edge weights [7]:

$$PR(u) = (1-c) + c \sum_{v \in B(u)} w(v,u) PR(v), \tag{2.6}$$

where $w(v,u)$ denotes the weight of the edge from node v to node u. Equation (2.5) can also easily be modified to support arbitrary weights by simply using a weighted adjacency matrix.

Solving Equation (2.5) might be inefficient for large networks due to the matrix inversion. However, we can improve the efficiency by using the power method. We therefore first have to reformulate Equation (2.4) such that:

$$\begin{aligned} r &= cAr + (1-c)\mathbf{1}qr \\ r &= (cAr + (1-c)\mathbf{1}q)r, \end{aligned} \tag{2.7}$$

where $\mathbf{1}$ denotes the vector of all ones. This step is valid since $\|r\|_1 = 1$ [5].

With $G = (cAr + (1-c)\mathbf{1}q)$ we get:

$$r = Gr, \tag{2.8}$$

which can be iteratively computed by [8]:

$$r^{(t+1)} = Gr^{(t)}, \tag{2.9}$$

for an arbitrary stochastic starting vector $r^{(0)}$, e.g. $r^{(0)} = q$. In order for (2.9) to converge, G has to be column-stochastic. For the rest of this thesis we will assume that a row-stochastic adjacency matrix A is given and we will therefore use the transposed adjacency matrix A^T instead of A.

2.1.2 Other Ranking Algorithms

PageRank is not the only ranking algorithm for networks. A similar measure which is also based on random walks is *Katz Centrality* [9]. The Katz measure sums over the collection of paths, where the path length (i.e. either the number of links in an unweighted network or the sum of weights in a weighted network) is exponentially discounted to favor shorter paths. Formally the Katz measure is defined as [10]:

$$K = (I - \beta A)^{-1} - I, \tag{2.10}$$

where β is a discount factor. The Katz measure is implicitly personalized such that K_{ij} denotes the importance of v_j if the random walk is started at node v_i.

Another ranking algorithm is *Hyperlink-Induced Topic Search (HITS)*, also known as *Hubs and Authorities*, which calculates two scores for each node, an *authority* and a *hub* score. The authority score measures the importance of the features of a node while the hub score measures the importance of outgoing links from a certain node [11]. Furthermore, there are some extensions to the original PageRank algorithm. One such extension is *AttriRank* which aims to incorporate node features into the ranking algorithm [6].

2.2 Spectral Clustering

Spectral clustering methods cluster data by using the eigenvectors of a matrix [12]. Since we are dealing with networks in this thesis, this similarity matrix will simply be the obtained weighted adjacency matrix $A \in \mathbb{R}^{n \times n}$. The number of clusters k needs to be know beforehand.

There are two different kinds of spectral clustering, namely *normalized* and *unnormalized* spectral clustering. We will only provide a short explanation of the unnormalized version. For more details and an explanation of the normalized version, please refer to Ulrike von Luxburg's tutorial on spectral clustering [13].

Unnormalized Spectral Clustering on graphs [13]:

1. Construct the Laplacian Matrix $L = D - A$, where D is the degree matrix.
2. Perform an eigenvalue decomposition: $L = U \Lambda U^T$
3. Construct a matrix V by taking the columns of U corresponding to the k smallest eigenvalues
4. Obtain n data points $y_1, \ldots y_n$, where y_i corresponds to the i-th row of matrix V
5. Cluster the obtained points y_i using a k-means algorithm into clusters $C_1, \ldots, C_k$

Spectral clustering essentially transforms the nodes of a graph and their respective pair-wise similarity scores into data points $y_1, \ldots, y_n \in \mathbb{R}^k$ such that they can be clustered using an algorithm like k-means.

Since spectral clustering methods heavily rely on a similarity matrix, we hope that we can achieve significantly better clustering results by using an adjacency matrix with learned weights instead of an unweighted adjacency matrix.

PageRank, and therefore all our weight learning approaches, work on directed graphs. However, spectral clustering assumes a symmetrical adjacency matrix. In order to perform spectral clustering on our learned networks, we need to either modify the clustering algorithm described above such that it can handle directed networks or we need to symmetrize the obtained adjacency matrix. We decided to use a symmetrization technique in order to apply spectral clustering on our learned networks. Our chosen symmetrization technique is to take the average of the sum of the adjacency matrix A and its transpose A^T. This corresponds to assigning an undirected edge between two nodes the average weight of the directed edges between them where a non-existent edge can be seen as an edge with weight zero. The symmetrized adjacency matrix A_{sym} is therefore defined as

$$A_{sym} = \frac{A + A^T}{2}. \tag{2.11}$$

Other symmetrization techniques can be found in [14]. Furthermore, a description of the alternative modification of the spectral clustering algorithm such that it works on directed networks can be found in [15].

2.3 Link Prediction

Link prediction techniques aim to infer which new connections are most likely to appear in the future given a snapshot of a current network [10]. These predictions are quite important in a variety of fields. Social networks could use it in order to suggest new friends to their users [16], companies might use it to recommend products to their customers [17] and researchers in the area of medicine or biology might utilize it to find promising relationships, e.g. in the area of protein-protein interaction [18].

We will now present a brief introduction to link prediction in social networks following the work of Liben-Nowell and Kleinberg [10].

In order to perform link prediction we need a graph $G = (V, E)$ where each edge $e \in E$ denotes a connection (e.g. collaboration, friendship or dependency) between two nodes and is labelled with a particular timestamp $t(e)$. Using these timestamps we can split the graph into a training set $G_{train} = (V, E_{old})$ and a test set $G_{test} = (V, E_{new})$ such that G_{train} contains all edges where $t(e) \in [t_0, t_1]$ and G_{test} all edges with $t(e) \in (t_1, t_2]$ with $t_0 < t_1 < t_2$. The set of nodes V is the same for both networks. It is possible that an edge between two nodes v_0, v_1 appears in both the training set and the test set, e.g. if two scientists collaborated before the splitting time t_1 and afterwards. Since link prediction concerns the prediction of new connections, all edges in G_{test} which also appear in G_{train} are removed such that all edges in G_{test} are connections which have not existed before t_1. It is now possible to pre-process the network and e.g. remove all nodes with too few edges in either G_{train} or G_{test}. A node v might for example have no edges in the training set because it has actually not been part of the network during the time $[t_0, t_1]$, maybe because user v of a social network was not a member of the network or employee v of a company had not worked for the company at that time. It is therefore impossible to predict links including this node v because it is unknown (i.e. no incoming connections) to all other nodes in the the training graph. Therefore it should be removed from both the training and the test set since it can only reduce the accuracy of a predictor. Liben-Nowell and Kleinberg set the threshold of removing nodes to 3 such that a node is removed if there are less than 3 adjacent nodes in either the training set or the test set.

A link prediction method is then essentially an algorithm which uses G_{train} to calculate a $\texttt{score}(x, y)$ for all node pairs $(x, y) \in V \times V$ which are not connected in the training set. This score can be seen as a measure of similarity between x and y and therefore corresponds to how likely a future connection between these two nodes is. These scores can then be used in order to predict links. One could either predict a link between all pairs (x, y) where $\texttt{score}(x, y)$ is larger than some threshold or by picking the n pairs with the highest scores. For evaluation purposes usually the latter method is used where n is set to the number of test edges $|E_{new}|$.

The score of a node pair can be calculated in several ways. We will give a brief overview over four common methods, further approaches can be found in [10].

- **Negated Shortest Path** [19]

  Negated Shortest Paths calculates the shortest path between all nodes in the network. These distance measures are then negated such that the shortest paths receive the highest scores.

- **Common Neighbors** [10]

  The common neighbors measure essentially follows the idea that two unconnected nodes are more likely to be connected in the future if they have many shared adjacent nodes. In a social friendship network this corresponds to common friends, where it is likely that two people will become friends if they know the same people. The `score` is formally defined as

  $$\texttt{score}(x, y) = |\Gamma(x) \cap \Gamma(y)|, \tag{2.12}$$

  where $\Gamma(x)$ denotes the set of adjacent nodes to x.

- **Katz Measure** [9]

  The Katz Measure essentially counts the paths from one node to another. In order to put more preference on shorter paths an exponential discount factor β is used. This yields the following `score` function

$$\texttt{score}(x, y) = \sum_{\ell=1}^{\infty} \beta^{\ell} \cdot |\text{paths}_{x,y}^{(\ell)}|, \tag{2.13}$$

where $|\text{paths}_{x,y}^{(\ell)}|$ denotes the number of paths from x to y using ℓ edges. In weighted networks the sum of weights is used instead of the number of paths. Equation (2.13) can be reformulated to a matrix equation which yields the Katz measure as described in Section 2.1.2:

$$K = (I - \beta A)^{-1} - I, \tag{2.14}$$

where K denotes the score matrix such that $S_{ij} = \texttt{score}(v_i, v_j)$ and A the adjacency matrix.

- **Personalized PageRank** [5, 10, 20]

  As described in Section 2.1.1, personalized PageRank is a measure of similarity between nodes. We can therefore get the scores for all nodes from node v_i by calculating the personalized PageRank where the query vector q is set to a unit vector consisting of all zeros and only q_i is set to one. More formally:

$$\texttt{score}(v_i, v_j) = ((1 - c)(I - cA^T)^{-1} q)_j \tag{2.15}$$

2.4 Related Work

In 2012 Bhat and Sims introduced an inverse PageRank approach in order to reproduce a network which has the same PageRank scores as a target ranking [21]. In essence, they started with a given adjacency matrix A and then tried to find a perturbation matrix $\Delta \in \mathbb{R}^{n \times n}$ such that

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \|P(A + \Delta) - \overline{P}\|, \tag{2.16}$$

where $P(A + \Delta)$ denotes the PageRank of the graph described by the adjacency matrix $A + \Delta$ and $\overline{P}$ denotes the desired PageRank. While the fundamental idea of finding a network which has a desired PageRank is similar to our objective, there are several differences to our approaches.

First, the perturbation matrix Δ might add edges to the network if $A_{ij} = 0$ and $\Delta_{ij} > 0$. This would change the structure of the graph. Our approaches instead learn weights and maintain the original graph structure. We furthermore introduce several extensions to Bhat and Sims approach. First of all, we consider personalized PageRank instead of a global PageRank score. Secondly, we introduce a method which can handle partial rankings where only the top k values of the target ranking are given. And third, we propose an algorithmic approach to learn edge weights even if just the ranking order is given.

To the best of our knowledge, we are the first to introduce methods to learn edge weights for cases where only the ranking order but no actual ranking scores are given.

Other related work regarding weight learning in networks mainly focuses on maximizing the PageRank score of one or more vertices instead of finding a weight assignment which achieves a desired PageRank [22, 23].

Part II

Network Learning via Ranking

Chapter 3

Ranking Value Loss

This chapter will present an algorithmic approach to learn edge weights for a given unweighted network and a given target ranking. The target ranking will in this case consist of numerical ranking scores. We will furthermore present an approach how weight learning is possible even if only a partial ranking consisting of the k most important nodes is given.

3.1 Problem Statement

The first problem we considered in the context of this thesis is how to find a weight vector w for a given unweighted network G_u such that its ranking $r(w)$ is equal to a given target ranking $\overline{r}$. More formally:

Problem 1
Let $G_u = (V, E)$ be a given unweighted network and Q a set of query vectors. Furthermore, a target ranking vector $\overline{r}_q$ describing the importance of each node $v \in V$ according to a specific ranking algorithm $r(w)$ is given for each query vector $q \in Q$. Then find a set of weights w, such that the weighted network G_w satisfies

$$r_q(w) = \overline{r}_q, \tag{3.1}$$

for every query vector $q \in Q$. From now on we define $n = |V|$ and $m = |E|$.

In the following, we will present an algorithmic approach for finding an approximate solution to Problem 1. We chose to use personalized PageRank (as described in Section

2.1.1) as our ranking method. However, the approach could easily be modified such that a different ranking algorithm is used.

3.2 Basic Approach

In order to learn a weight vector w we first recast the problem statement as a minimization problem since it might be hard to find an exact solution. We therefore define our optimal weights w^* and our objective function $f(w)$ as:

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^m} f(w) \tag{3.2}$$

$$f(w) = \sum_{q \in Q} \frac{1}{2} \|\overline{r}_q - r_q(w)\|_2^2. \tag{3.3}$$

Since we chose PageRank as our ranking algorithm r, we have to consider some constraints when solving Equation (3.2). First of all, the ranking output of PageRank are values $r_{q,v}$, where $r_{q,v} \geq 0$ and $\|r_q\|_1 = 1$ for every query vector $q \in Q$ and every node $v \in V$. We therefore assume that the target ranking $\overline{r}_q$ satisfies these constraints as well. Furthermore, it is necessary that each node v has a weighted out-degree of 1 (i.e. the weights of outgoing edges sum up to 1) and that each edge weight is positive. These constraints lead to an updated formulation of Equation (3.2)

$$\begin{aligned} f(w) = & \sum_{q \in Q} \frac{1}{2} \|\overline{r}_q - r_q(w)\|_2^2 \\ \text{s.t.} \quad & r_q(w) = (1-c)\left(I - cA(w)^T\right)^{-1} q \quad, \forall q \in Q \\ & \sum_{j=1}^{n} A(w)_{ij} = 1, \quad, \forall i \in [n] \\ & w_i > 0, \quad, \forall i \in [m], \end{aligned} \tag{3.4}$$

where $A(w)$ denotes the adjacency matrix of the network with weights w and A^T the transpose of A.

To find an optimal solution w^*, we chose an iterative gradient-based optimization method. Therefore, we need to compute the gradient of our objective function f with respect to w. We obtain the gradient by

$$\begin{aligned}\nabla_w f(w) = \nabla_w \left(\sum_{q \in Q} \frac{1}{2} \|\overline{r}_q - r_q(w)\|_2^2 \right) = \\ = \sum_{q \in Q} \frac{1}{2} \nabla_w \|\overline{r}_q - r_q(w)\|_2^2 = \sum_{q \in Q} \nabla_w \left(r_q(w) \right) \left(\overline{r}_q - r_q(w) \right)\end{aligned} \tag{3.5}$$

By implicitly differentiating[1] the PageRank equation (2.5), we obtain

$$\nabla_w (r_q(w)) = c(I - cA(w)^T)^{-1} \nabla_w (A(w)^T) r_q(w). \tag{3.6}$$

$\nabla_w(A(w)^T)$ is hereby the gradient of matrix $A(w)^T$ with respect to the vector w which would be a vector of matrices such that $\nabla_w(A(w)^T) \in \mathbb{R}^{m \times n \times n}$. Multiplying this with $r_q(w)$ results in a two-dimensional $\mathbb{R}^{m \times n}$ matrix. More formally, this gradient can be defined as:

$$\nabla_w (A(w)^T) r_q(w) = \begin{pmatrix} \frac{\partial A(w)^T}{\partial w_0} r_q(w) & \dots & \frac{\partial A(w)^T}{\partial w_m} r_q(w) \end{pmatrix}, \tag{3.7}$$

where

$$\left(\frac{\partial A(w)^T}{\partial w_k} \right)_{ij} = \begin{cases} 1, & \text{if } (A(w)^T)_{ij} = w_k \\ 0, & \text{else} \end{cases} \tag{3.8}$$

Now, inserting (3.7) into (3.5) yields the complete gradient of our objective function:

$$\sum_{q \in Q} (I - cA(w)^T)^{-1} c \nabla_w (A(w)^T) r_q(w) (\overline{r}_q - r_q(w)). \tag{3.9}$$

3.3 Top-K

Our approach presented in Section 3.2 requires that every element of the target ranking $\overline{r}_q$ is known. However, in some cases only the most important nodes and their respective scores might be known for a given query vector q. We therefore developed a modification to our full-ranking approach which can handle this top-k scenario.

[1]The full implicit differentiation computation can be found in Appendix A

PageRank is an algorithm which always calculates the ranking score of every node in the network. Since in the top-k scenario now only the k highest ranking scores are given for a specific query vector, we need to modify our objective function f (see Equation (3.4)) such that our computed ranking scores for nodes which are not included in the k most important nodes do not affect the objective since there is no target value for these nodes. The target ranking score for the $n-k$ nodes not included in the k most important nodes can be assumed to be 0. We can therefore simply set the computed elements of the PageRank vector r_q to 0 such that only the differences of the k most important nodes are considered in our objective.

Our adapted objective function is therefore defined as:

$$\begin{aligned} f(w) &= \sum_{q \in Q} \frac{1}{2} \|\overline{r}_q - top_k(r_q(w))\|_2^2 \\ \text{s.t.} \quad & r_q(w) = (1-c)\left(I - cA(w)^T\right)^{-1} q \quad ,\forall q \in Q \\ & \sum_{j=1}^{n} A(w)_{ij} = 1, \quad ,\forall i \in [n] \\ & w_i > 0, \quad ,\forall i \in [m], \end{aligned} \tag{3.10}$$

where $top_k(r_q(w))$ sets the ranking scores of all nodes which do not belong to the k most important nodes in the target ranking to zero.

The modification of the objective function gradient (3.9) can be done analogously. Our current approach assumes that k is equal for all nodes. However, the extension to independent values k_i would be trivial.

Chapter 4

Ranking Order Loss

In Learning to Rank one can either try to find a function such that each item is assigned a specific ranking score or a function which puts the items in a specific order. In Chapter 3 we already described an approach how to assign a specific PageRank score to each node in a given network. In this chapter, we will now present two different approaches to find edge weights such that the ranking order of nodes in a network equals a given target order.

4.1 Problem Statement

The problem we are trying to solve in this chapter differs from our previous problem (see Section 3.1 Problem 1). In the value loss setting the exact ranking scores for all or just some nodes with respect to a certain query vector were given. Now, we only know a target ranking order $\overline{R}$, where $\overline{R}$ is a permutation of $[n]$ such that $\overline{R}_i$ describes the position of node v_i in the ranking order. More formally:

Problem 2
Let $G_u = (V, E)$ be a given unweighted network and Q a set of query vectors. Furthermore, a target ranking order $\overline{R}_q$ is given, where $\overline{R}_{q,i} \in [n]$ represents the position of node v_i in the ranking. Hereby the most important node has importance 1, the second most important node 2, and the least important node n.

Then find a set of weights w, such that the weighted network G_w satisfies

$$R_q(w) = \overline{R}_q, \tag{4.1}$$

for every query vector $q \in Q$. Where R_{qi} denotes the position of node v_i in the computed PageRank vector r_q.

4.2 Naïve Approach

Our first approach attempts to recast Problem 2 to Problem 1 such that we can use the methodology described in Section 3.2. Therefore, we need to convert the given target ranking order $\overline{R}_q$ to PageRank scores.

Such a conversion can be done in multiple ways. The easiest way would be to use a linear function to assign each node a ranking score. One such linear function could be:

$$\overline{r}_{q,i} = \frac{n - \overline{R}_{q,i}}{\sum_{i=1}^{n} i}, \tag{4.2}$$

where the division by $\sum_{i=1}^{n} i$ is necessary since $\overline{r}_q$ has to be a stochastic vector. Figure 4.1 shows an example of such a conversion.

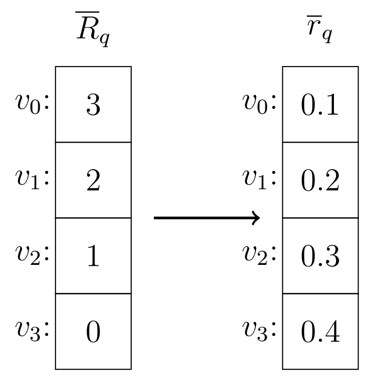

Figure 4.1: Linear Mapping to convert a ranking order $\overline{R}_q$ to a PageRank vector $\overline{r}_q$.

However, such linear mappings do not represent most real-life networks in which there are usually just a few important nodes and many nodes with lower importance scores. We therefore used exponential decay to convert the target order to scores instead of a linear mapping. Hence, our conversion function is defined as

$$\overline{r}_{q,i} = \frac{e^{-\lambda \overline{R}_{q,i}}}{\sum_{i=1}^{n-1} e^{-\lambda i}}, \tag{4.3}$$

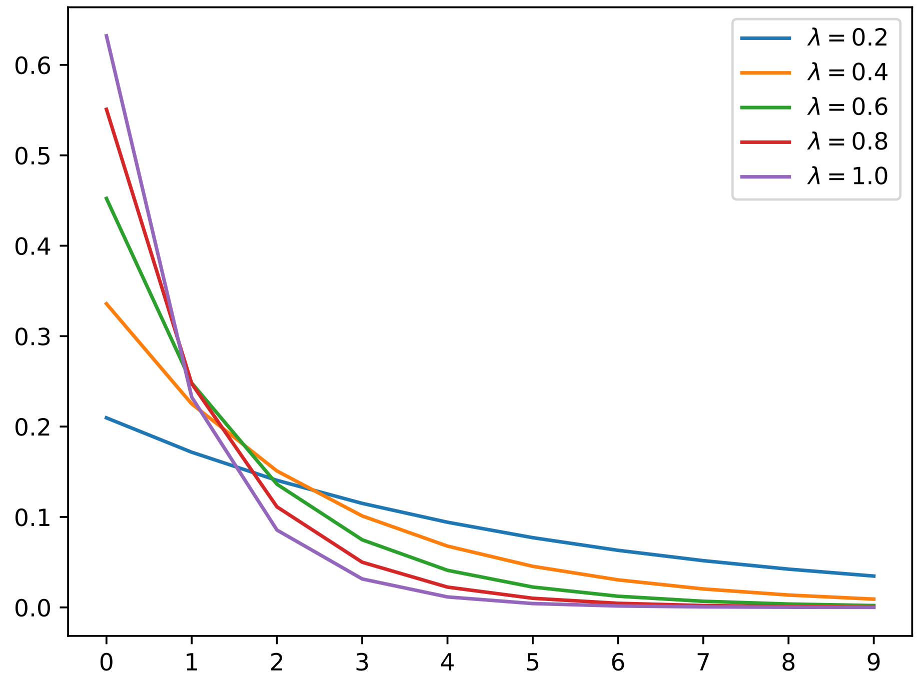

Figure 4.2: Ranking Scores for 10 Nodes converted using Exponential Decay with different values of λ.

where $\sum_{i=}^{n-1} e^{-\lambda i}$ is again used for normalization and $\lambda > 0$ is the exponential decay constant.

The choice of λ affects how strong the ranking scores decrease for higher positions (i.e. lower importance). Figure 4.2 shows the ranking score distribution for different choices of λ for a network consisting of 10 nodes. Essentially, there are two ways of choosing λ. One can either fix λ to a certain value like $\lambda = 0.1$ or it can be adapted based on the number of nodes in the network, e.g. $\lambda = \frac{10}{n}$. If a fixed value is chosen the same number of top nodes always receives a higher ranking before the ranking scores converge to zero. Ranking order conversions for $\lambda = 0.1$ can be seen in Figure 4.3 while conversions for $\lambda = \frac{10}{n}$ are shown in Figure 4.4. For the remainder of this thesis we will assume $\lambda = 0.1$.

After we converted the given target order to corresponding target scores, we can now use the approach described in Section 3.2 in order to find edge weights. If we minimize the value loss objective (3.4), we will also achieve that our computed ranking order is equal to our target ranking order.

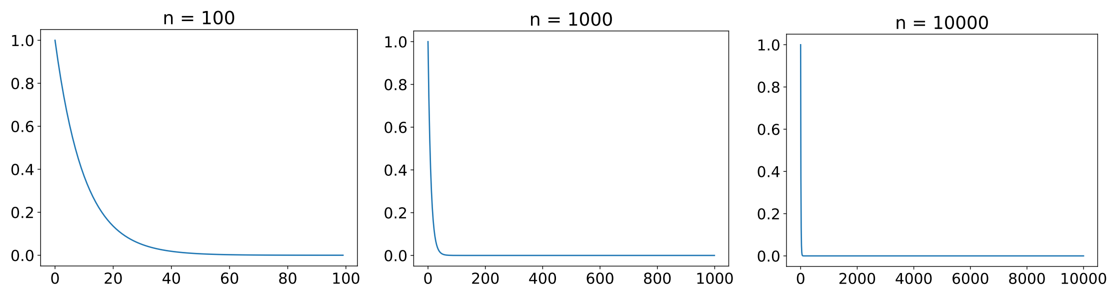

Figure 4.3: Ranking Scores for $\lambda = 0.1$ for different graph sizes.

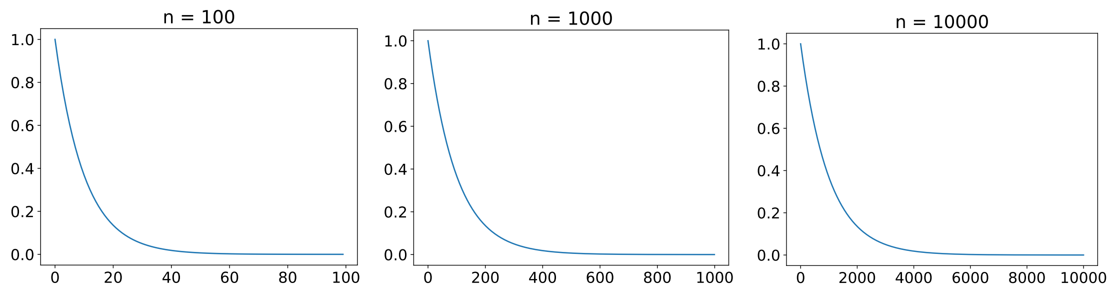

Figure 4.4: Ranking Scores for $\lambda = \frac{10}{n}$ for different graph sizes.

4.3 Hinge Loss Approach

Additionally, we also developed an approach which does not require a transformation from the target order to ranking scores but instead uses the target order directly. In the interest of finding a set of weights, such that the computed ranking order is similar to the given target order, we would ideally directly minimize a ranking measure such as Spearman's Footrule, Average Precision or (Normalized) Discounted Cumulative Gain (see Section 7.1). However, these ranking measures are not continuous and are therefore hard to optimize. Hence, we defined a continuous objective function which considers the ranking positions of all nodes:

$$f(w) = \sum_{q \in Q} \sum_{i=1}^{n} \sum_{j=1}^{n} \mathcal{L}(i,j), \tag{4.4}$$

where $\mathcal{L}(i,j)$ is a pairwise asymmetric Hinge-Loss defined as:

$$\mathcal{L}(i,j) = \mathbb{1}(\overline{R}_{q,i} < \overline{R}_{q,j}) \max\left(0, r_q(w)_j - r_q(w)_i\right), \tag{4.5}$$

where $\mathbb{1}$ denotes the indicator function and $r_q(w)_i$ the i-th element of $r_q(w)$.

The combined pairwise loss $\mathcal{L}(i,j) + \mathcal{L}(j,i)$ is therefore 0, if the computed ranking score of the more important target node is higher than the computed ranking score of the other node which essentially is the case if nodes v_i and v_j are in the correct order. Otherwise $\mathcal{L}(i,j) + \mathcal{L}(j,i)$ is the difference between their ranking scores.

The gradient of our objective function f is defined as

$$\sum_{q \in Q} \sum_{i=1}^{n} \sum_{j=1}^{n} \nabla_w \mathcal{L}(i,j) = \sum_{q \in Q} \sum_{i=1}^{n} \sum_{j=1}^{n} \begin{cases} \nabla_w (r_q(w)_j) - \nabla_w (r_q(w)_i), \text{ if } \overline{R}_{q,i} < \overline{R}_{q,j} \\ 0, \text{ else} \end{cases} , \quad (4.6)$$

where $\nabla_w(r(w))$ is defined as in (3.6).

4.3.1 Hinge-Loss with Margin

One possible problem of Equation (4.4) is that the total loss would be very small if all nodes got about the same ranking even if the ranking order is quite different from the target order. Furthermore, if the algorithm manages to achieve a perfect ranking, the ranking scores might be almost equal for all nodes. We therefore introduced a margin into the loss function.

$$\mathcal{L}(i,j) = \mathbb{1}(\overline{R}_{q,i} < \overline{R}_{q,j}) \max \left(0, r_q(w)_j - r_q(w)_i + \frac{1}{n} \right). \quad (4.7)$$

This margin helps to increase the distance between ranking scores of neighbouring positions. Assuming that $\overline{R}_{q,i} < \overline{R}_{q,j}$, the loss function $\mathcal{L}(i,j)$ would only be zero, if $r_q(w)_i > r_q(w)_j + \frac{1}{n}$.

4.3.2 Position-Aware Hinge Loss

Another extension to our original Hinge-Loss approach (see Equation (4.4)) is to include a position-dependent factor. The essential idea is that it should be stronger penalized if a very unimportant node is ranked higher than an important node. Hence, we weighted the pairwise Loss $\mathcal{L}(i,j)$ in Equation (4.4) by the absolute difference of $\overline{R}_{q,i}$ and $\overline{R}_{q,j}$ such that our new objective is defined as

$$f(w) = \sum_{q \in Q} \sum_{i=1}^{n} \sum_{j=1}^{n} |\overline{R}_{q,i} - \overline{R}_{q,j}| \mathcal{L}(i,j), \tag{4.8}$$

4.4 Top-K

As for the value-loss approach (see Section 3.3) we also developed a solution for cases where only the ordering of the k most important nodes is known.

4.4.1 Naïve Approach

Since in our first approach (see Section 4.2) we converted the target ranking order into ranking scores, it is simply possible to use the same method as described in Section 3.3 in order to handle the top-k scenario.

4.4.2 Hinge-Loss

In order to modify our Hinge-Loss approach (see Equation (4.4)) such that it is able to handle the top-k scenario, we need to simply change the iterator of the second sum such that only nodes $v_i \in top_k(\overline{R}_q)$ are considered. Our updated objective function is therefore defined as

$$f(w) = \sum_{q \in Q} \sum_{i : v_i \in top_k(\overline{R}_q)} \sum_{j=1}^{n} \mathcal{L}(i,j). \tag{4.9}$$

Part III

Evaluation and Conclusion

Chapter 5

Setup

Before presenting the actual results of our evaluation, we first describe how we conducted our experiments. We will start by describing the networks we used for testing. Furthermore we will provide some details about our implementation.

5.1 Datasets

In order to test and compare our approaches, we collected several real-world datasets and additionally created synthetic datasets. Most real-world dataset are not in compliance with the requirements of PageRank (i.e. out-going edges sum up to 1 for all nodes). We therefore had to pre-process the networks by scaling the edge weights and removing isolated nodes. Our used datasets and the necessary pre-processing modifications are described below.

1. **Barabási-Albert-Networks** [24]
   Barabási-Albert-Networks are synthetically generated random scale-free networks. In consequence, their degree distribution follows a power law.
   The creation algorithm takes two arguments: the number of nodes n and a degree factor λ. It then starts with λ initial nodes and arbitrary connections between them such that each initial node is connected. Then in each step, a new node is added and connected to m random nodes in the existing network. The probability $\pi(v_i)$ of connecting the new node to node v_i is hereby defined as

$$\pi(v_i) = \frac{k_i}{\sum_j k_j}, \tag{5.1}$$

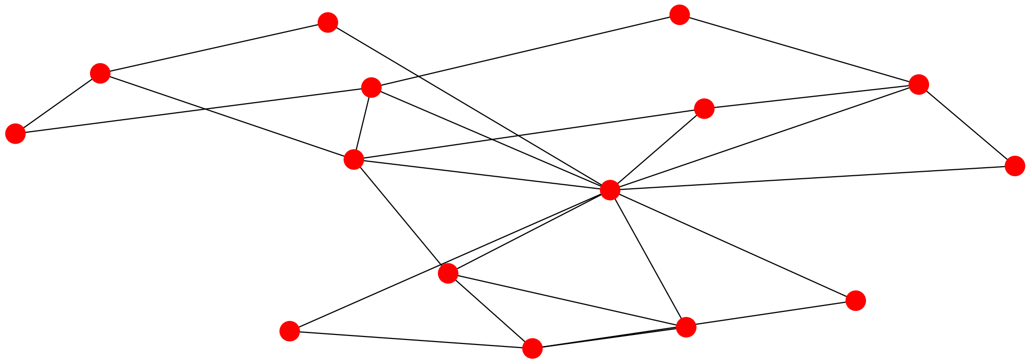

Figure 5.1: Barabási-Albert-Network with $n = 15$ and $\lambda = 2$

where k_i is the degree of node v_i. It is therefore more likely, that a new node is connected to a node with a higher degree. Figure 5.1 shows an example of a Barabási-Albert-Network. Barabási-Albert-Networks are undirected by default. In order to obtain a directed network, we simply replaced each undirected edge by two directed edges. Since we furthermore need a weighted network, we draw each edge weight from a uniform distribution and then rescaled the weights to comply with the PageRank requirements. In the following chapters we often use two Barabási-Albert-Networks, a small one with $n = 1000$ and $\lambda = 2$ consisting of 3992 edges and a larger one with $n = 5000$ and $\lambda = 5$ consisting of 49950 edges.

2. **Cora** [25]
   The Cora dataset consists of 2708 nodes which represent scientific publications categorized into seven classes. Each of these publications is described by a binary vector indicating whether a certain word from a dictionary of 3703 words appeared in the publication or not. The dataset furthermore contains 5278 directed edges where an edge from one node to another means that the first publication cited the second one. We however used an undirected version of Cora such that our used network consisted of 5278 undirected or equivalently 10556 directed edges.

   After pre-processing the original Cora dataset, we obtained a network of 2586 nodes and 8910 edges. In order to use Cora for our approach, we also needed to find edge weights for the network. We therefore defined the weight w_{ij} of an edge from node v_i to node v_j as the cosine similarity between the binary vectors of both publications.

3. **Last.fm Dataset (Subset)** [26]
   The Last.fm song similarity subset contains a similarity network consisting of 9330 nodes. After pruning all nodes which do not have any similar nodes in the subset

we obtained a network of 3091 nodes and 7,318 edges.

4. **Netscience** [27]
   A node in the Netscience dataset represents a network theory scientist. An edge between two scientists represents a co-authorship between these scientists. The edge weights are calculated as in [28]. After removing all isolated nodes, we obtained a network of 1461 nodes and 5484 edges.

5. **Advogato** [29, 30]
   Nodes of this dataset represent users of the online community platform Advogato. An edge from user v_1 to user v_2 represents the trust of v_1 in v_2. In contrast to the other datasets we used, edge weights of the Advogato network are discrete, such that $w \in \{0.6, 0.8, 1.0\}$ for all edge weights w. After removing all loops and isolated nodes, we obtained a network consisting of 3830 nodes and 42519 edges.

6. **Primary School Contacts** [31, 32]
   This dataset represents contacts between children in a primary school. Each node represents a child and an edge between node v_i and v_j denotes that v_i and v_j had contact during the measurement period where the edge weight represents the number of contacts between the two people. Additionally this dataset contains for each node to which school class it belongs. The original dataset actually contained information about teachers as well. However, we removed the teacher nodes from the network since we will use this dataset for the spectral clustering application in Chapter 8 and the teacher class is rather small which might negatively impact the performance of a clustering algorithm. After pre-processing we obtained a network of 232 nodes and 15712 edges.

7. **UC Irvine Communication** [33, 34]
   This network represents students from the University of California, Irvine and messages sent between them. The edge weight denotes how many messages had been sent between two students. Additionally each edge contains timestamps denoting the times each single message has been sent. In total there are 1899 students and 27676 connections in the network. The messages were all sent between 15th April 2004 and 26th October 2004.

8. **DBLP Collaboration** [35, 36]
   The DBLP Collaboration dataset represents scientific authors from the DBLP computer science bibliography. An edge between two authors denotes the number of common publications. Following [37] we split the original dataset into 4 sub-networks representing the research areas Information Retrieval (IR), Data Mining (DM),

Machine Learning (ML) and Databases (DB). More information about node counts and edges will be provided in Chapter 9.

A table containing more detailed statistics of each dataset can be found in Appendix B.

5.2 Implementation

We implemented our approach in Python. In order to solve the minimization problems, we used the large-scale interior-point solver IPOPT [38][1] combined with a linear solver obtained from HSL [39].

We note for reference that we ran all experiments on a computer with a 2.3 GHz Intel Core i7 and 16GB RAM.

[1]Eric Xu's PyIpopt Implementation was used in order to access IPOPT using Python. Github-Repository: `https://github.com/xuy/pyipopt`

Chapter 6

Ranking Value Loss

In this chapter we present the results of several numerical experiments proving the feasibility of our approach to recover edge weights for given ranking scores.

In the first section we will analyse the scalability of our algorithm. Afterwards, we show how accurately our algorithm is able to recover edge weights for several datasets.

6.1 Scalability

The scalability of our algorithm depends on several factors. First of all, the network itself has a high impact on the necessary cost. In order to evaluate how our algorithm scales, we ran our algorithms several times while increasing the number of edges. We first conducted this experiment on synthetic datasets creating two Barabási-Albert-Networks with a fixed number of nodes ($n = 2000$ and $n = 4000$) and different values of λ to increase the number of edges. Additionally, we repeated the experiment using the real-world network "Advogato". To control the number of edges in this case, we subsampled the network by only using the first n nodes in the beginning and then increasing n for the next measurements. The resulting runtimes are displayed in Figure 6.1. It can be seen that the number of edges has a linear impact on the runtime. Hence, our value loss approach is efficient enough to handle weight learning in large-scale networks. The impact of the amounts of nodes on the runtime can be seen by comparing the plots for the two synthetic Barabási graphs. The runtime does not differ by much although one graph has twice as many nodes as the other.

Besides the edges, the quantity of query vectors is a relevant factor for the time complexity

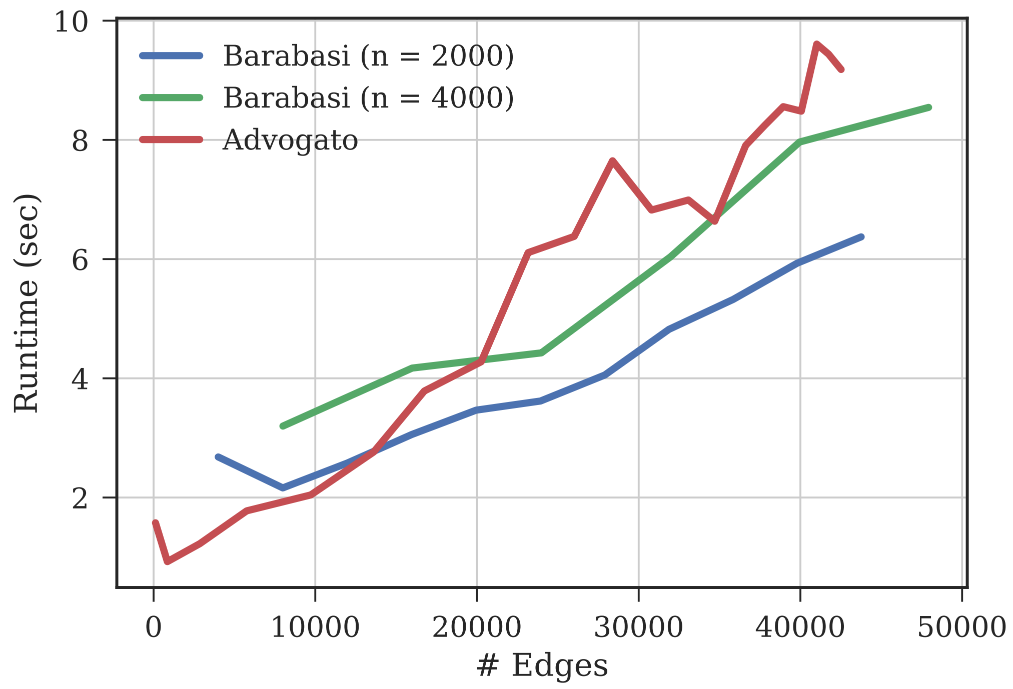

Figure 6.1: Influence of the edge count on the runtime. The algorithm was run for 100 iterations using one randomly chosen query vector.

of our algorithm. It can be seen from Equation (3.4) and Equation (3.9), that using more query vectors increases the number of terms in the summation. Computing the result of each of these terms will take roughly the same amount of time, since they consist of the same sequence of operations. Hence, the impact of query vectors on the total runtime of our algorithm is just linear. However, since the computations for each query vector are independent from the others, we can parallelize their computation and therefore further reduce the time complexity. Figure 6.2 shows the time our algorithm needed to complete 100 iterations for different amounts of query vectors for the Advogato network and a small Barabási-Albert-Network ($n = 1000, \lambda = 2$). Furthermore, the respective runtimes of our parallelized implementation are shown.

The performance of the top-k approach (see Section 3.3) is roughly the same as the normal approach since we simply set the computed ranking scores for non-top-k nodes to zero, while all other steps are identical.

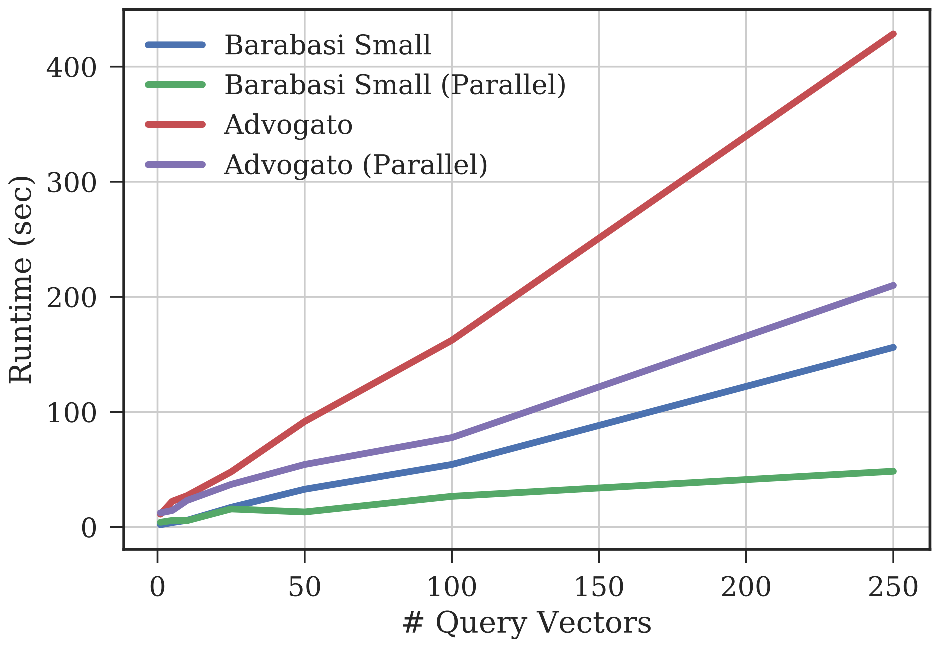

Figure 6.2: Runtime Impact Query Vectors in a parallel (8 threads) or non-parallel implementation (100 iterations).

6.2 Accuracy

After analysing the efficiency of our approach, we will now present the results of numerical experiments regarding its accuracy. We will start by showing that our implementation indeed minimizes our objective function f and then present the quality of our recovered weights.

6.2.1 Optimization

While we learn our edge weights by minimizing a loss function over the computed and target ranking scores, the actual goal of this method is to recover the edge weights of the network. Hence, we first verified that our objective function really converges by plotting the result of our objective function after each iteration. We conducted this convergence test for the Last.fm and the Cora dataset. The proof that our objective indeed does converge to zero can be seen in Figure 6.3. This test furthermore showed that our approach converges quite fast. It is therefore not necessary to run the algorithm for many iterations.

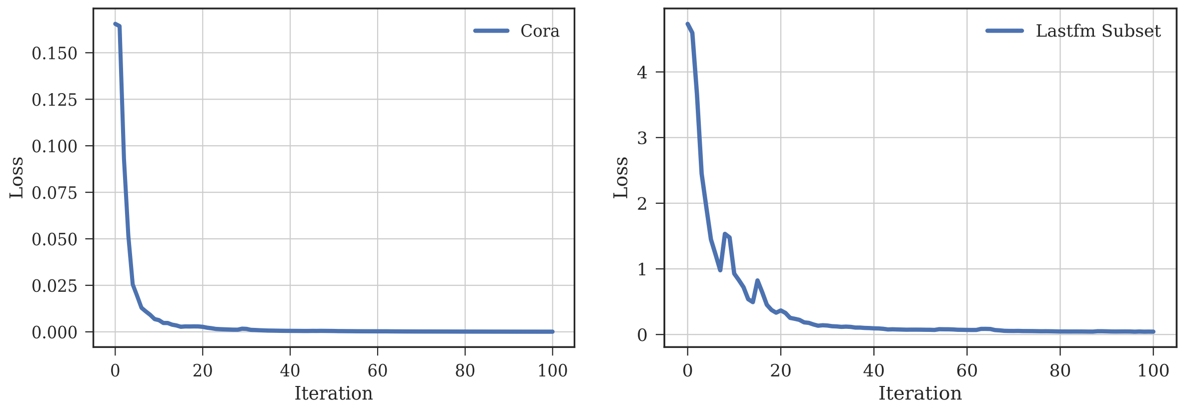

Figure 6.3: Convergence of the objective function for 100 iterations using 100 query vectors.

6.2.2 Recovering Quality

Now that we have seen that our loss function actually converges to zero, we also need to check whether the obtained weights correspond to the original weights. We will start by analysing the recovering quality of our full ranking approach and then repeat this for our top-k approach.

6.2.2.1 Full Ranking Approach

In order to determine the weight recovering quality of our full ranking approach we ran our algorithm for 100 iterations on several of the datasets described in Section 5.1 while varying the number of randomly chosen query vectors. We evaluated the results of our experiments based on following measurements:

- **Mean Weight Distance** (MWD)
  This measure essentially tells us how close our learned weights are in average to the original weights of the network. We therefore calculated the mean absolute distance (L1) the mean Euclidean distance (L2) between learned and original weights.
- **Mean Ranking Distance** (MRD)
  This measure describes how close to the target ranking our algorithm could get in average. We again computed the mean absolute distance (L1) and the mean Euclidean distance (L2).
- **Runtime** (RT)
  In order to compare the results of runs with different amounts of query vectors we also have to take into account how much time it took to achieve a result.

The result of these tests are shown in Table 6.1. It is interesting to see that the mean ranking distance (MRD (L1) and MRD (L2)) is usually smaller if less query vectors are used. Since more query vectors implicate that more target rankings have to be fit, this result essentially means that it is easier to find weights such that fewer rankings are fit than more rankings. However, Table 6.1 also shows, that in most cases the weight distances are better for more query vectors. This indicates, that a given ranking for a query vector does not completely determine all weights of the graph. Instead, apparently there exist several edge weight assignments such that the same ranking scores are achieved. This assumption is confirmed by the results of the Last.fm dataset, where it can be seen that our algorithm is able to find quite similar ranking scores but the weights differ on average by more than 0.10 from the original weights. For most other cases, we are able to obtain weights which on average have an absolute difference of less than 0.05 from the original weights. This difference could probably be even further improved by running the algorithm for more iterations and with more query vectors. However, this would also increase the total runtime. A very interesting case is the Advogato dataset, where our algorithm is able to find almost identical weights even for a small quantity of query vectors.

As we have shown, our algorithm is able to learn edge weights such that the computed ranking is almost equal to a given target ranking. Furthermore, the learned weights correspond closely to the original weights for most datasets.

	$\lvert Q\rvert$	MWD (L1)	MWD (L2)	MRD (L1)	MRD (L2)	RT (sec)
Barabasi Small	1	0.110	**0.002**	**3.7e-06**	**4.7e-11**	**5.54**
	10	0.105	0.003	1.7e-04	1.5e-07	5.91
	50	0.067	0.003	5.4e-05	1.3e-06	53.68
	100	**0.049**	**0.002**	7.6e-05	3.2e-08	76.97
Barabasi Large	1	0.064	4.4e-04	**1.2e-05**	**2.1e-10**	**8.54**
	10	0.063	4.4e-04	2.5e-05	1.5e-09	102.23
	50	0.058	4.1e-04	2.7e-05	0.3e-08	224.90
	100	**0.052**	**3.8e-04**	2.4e-05	2.2e-09	278.38
Advogato	1	0.006	6.2e-05	**1.6e-06**	**8.0e-12**	**28.19**
	10	0.006	6.6e-05	1.8e-06	1.9e-11	59.51
	50	0.006	6.5e-05	5.1e-06	3.1e-10	35.34
	100	**0.005**	**6.0e-05**	1.8e-06	2.2e-11	211.37
Cora	1	0.052	8.3e-04	**2.9e-06**	**2.3e-11**	**8.34**
	10	0.056	9.4e-04	1.2e-05	4.2e-10	31.74
	50	0.033	6.9e-04	5.0e-06	1.0e-10	75.04
	100	**0.030**	**6.5e-04**	6.1e-06	2.1e-10	102.02
Last.fm	1	0.106	**0.002**	2.0e-05	**1.9e-09**	**8.54**
	10	0.122	**0.002**	4.6e-05	1.1e-06	61.80
	50	0.114	**0.002**	**1.3e-05**	2.0e-08	82.00
	100	**0.105**	**0.002**	**1.3e-05**	2.6e-08	134.10
Netscience	1	0.025	**6.4e-04**	**5.2e-07**	**3.4e-12**	**2.88**
	10	0.028	8.6e-04	8.0e-06	6.6e-10	23.10
	50	0.025	7.8e-04	2.3e-06	7.0e-10	54.99
	100	**0.021**	6.6e-04	4.5e-06	2.8e-09	81.79

MWD: Mean Weight Distance, MRD: Mean Ranking Distance, RT: Runtime

Table 6.1: Obtained Results on several datasets using our value loss approach. The algorithm was run for 100 iterations with different amount of query vectors ($\lvert Q\rvert$). Best results are marked in bold.

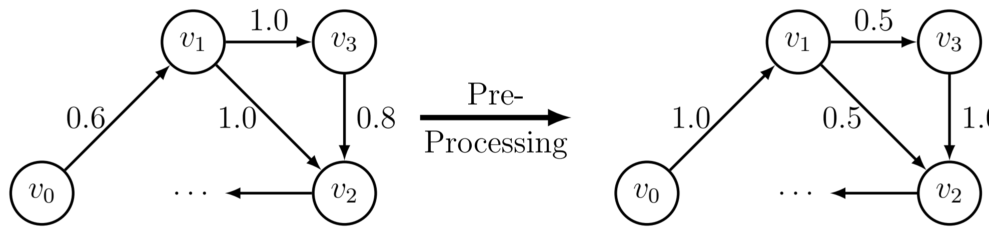

Figure 6.4: Pre-processing of a subgraph of the Advogato network. It can be seen that the meaning of edge weights is changed by the scaling. In this Figure "..." represents the rest of the network

6.2.2.2 Top-K Approach

It now remains to be seen, how our algorithm can handle the top-k scenario as described in Section 3.3. To examine the effectiveness of our method, we varied the value of k and therefore the amount of given ranking scores while fixing the number of used query vectors to 10 and the iterations to 100. The results of this experiments are displayed in Table 6.2. Please note that $k = n$ denotes that we set k to the total number of nodes in the respective network such that a full ranking is given. The results show that our top-k approach is capable of recovering weights almost as good as if a full ranking was given. We are therefore able to handle networks where only a partial ranking is given for each node.

Thus far we have shown that our approach is able to learn edge weights for both synthetic and real-world datasets. However, it is important to notice that we were only able to recover the *pre-processed* edge weights. As mentioned in Section 5.1, it is necessary to rescale all weights such that $w_i \in (0, 1]$ and the outgoing edges sum up to 1 for each node. This rescaling process unfortunately might change the meaning of an edge weight. Suppose we pre-process the edge-weights of the Advogato dataset where $w_i \in \{0.6, 0.8, 1.0\}$ for all edge weights w_i. Figure 6.4 shows a possible small subnetwork of 4 users and their trust relationships and the corresponding pre-processed network. It can be seen that weights of edges from nodes with only one outgoing edge (v_0, v_3) are all rescaled to 1.0 regardless of their original value. Furthermore, equally weighted edges originating in the same node (e.g. v_1) will still have equal weight after the pre-processing step, however, the meaning of these edges is lost as well, since it is impossible to determine whether their original weight was 0.6, 0.8 or 1.0. Our approach is therefore unable to recover the original, un-processed weights, unless we know the original sum of outgoing weights of each node. Nevertheless, the normalized edge weights might still be able to improve the performance graph algorithm like clustering or link prediction compared to the performance using an unweighted network.

	k	MWD (L1)	MWD (L2)	MRD (L1)	MRD (L2)	RT (sec)
Barabasi Small	10	1.30e-01	2.88e-03	5.21e-04	1.43e-06	12.49
	25	1.24e-01	2.78e-03	3.79e-04	6.72e-07	10.10
	50	1.28e-01	2.84e-03	4.37e-04	5.24e-07	**6.84**
	100	1.29e-01	2.86e-03	3.68e-04	3.51e-07	8.17
	n	**1.21e-01**	**2.76e-03**	**1.46e-04**	**8.13e-08**	11.93
Barabasi Large	10	6.69e-02	4.60e-04	1.29e-04	8.31e-08	**28.70**
	25	6.67e-02	4.59e-04	1.25e-04	4.93e-08	37.07
	50	6.62e-02	4.57e-04	1.12e-04	3.15e-08	43.59
	100	6.57e-02	4.54e-04	9.89e-05	2.03e-08	38.12
	n	**6.48e-02**	**4.50e-04**	**3.35e-05**	**3.68e-09**	34.16
Advogato	10	7.02e-03	6.78e-05	1.61e-04	2.72e-07	27.15
	25	6.98e-03	6.76e-05	1.37e-04	1.44e-07	24.41
	50	6.98e-03	**6.73e-05**	1.02e-04	7.30e-08	22.12
	100	7.05e-03	6.78e-05	9.17e-05	6.01e-08	29.65
	n	**6.94e-03**	6.75e-05	**1.55e-05**	**5.91e-09**	**15.10**
Cora	10	7.37e-02	1.13e-03	2.29e-04	4.58e-07	25.05
	25	7.22e-02	1.12e-03	1.96e-04	1.95e-07	13.03
	50	6.97e-02	1.09e-03	1.58e-04	9.84e-08	14.09
	100	6.90e-02	1.09e-03	1.34e-04	5.76e-08	14.72
	n	**6.21e-02**	**1.00e-03**	**1.85e-05**	**1.13e-09**	**11.13**
Last.fm	10	1.26e-01	2.37e-03	9.25e-05	9.64e-07	13.44
	25	1.25e-01	2.36e-03	6.84e-05	7.68e-07	12.17
	50	1.25e-01	2.36e-03	4.32e-05	2.92e-07	**9.73**
	100	**1.21e-01**	**2.31e-03**	2.05e-05	5.24e-08	13.23
	n	1.23e-01	2.33e-03	**1.64e-05**	**2.99e-08**	11.29
Netscience	10	2.86e-02	8.76e-04	5.87e-05	1.24e-06	**9.12**
	25	2.80e-02	8.70e-04	7.72e-06	2.07e-08	10.85
	50	2.81e-02	8.60e-04	9.59e-06	1.18e-08	12.11
	100	**2.76e-02**	**8.53e-04**	1.19e-05	8.00e-09	23.23
	n	2.80e-02	8.68e-04	**3.14e-06**	**1.42e-09**	14.38

MWD: Mean Weight Distance, MRD: Mean Ranking Distance, RT: Runtime

Table 6.2: Obtained Results on several datasets using our value loss approach. The algorithm was run for 100 iterations with different top-k values with 10 query vectors. Best values are marked in bold.

6.3 Conclusion

In this chapter, we have analysed the efficiency and effectiveness of our approach. We have shown that we are able to learn edge weights for a given network for given target PageRank vectors. We noticed, that it is possible to achieve a good accuracy even with only a few query vectors. Furthermore, edge weights for networks where only a partial ranking consisting of the k most important ranking scores for each query vector is given, can be recovered almost as good as if a full ranking was given.

However, we had to notice that the necessary pre-processing of the network might destroy some information. In order to solve this problem of recovering similarity scores, the weighted out-degree of every node would have to be known.

Chapter 7

Ranking Order Loss

This chapter will present an analysis of our two ranking order loss approaches, the conversion from a ranking order to ranking scores and the pairwise ranking score Hinge loss. We start by describing several measures how the compliance of two ranking orders can be quantified. Afterwards an evaluation of the scalability and accuracy of our approaches will be shown.

7.1 Ranking Measures

In order to determine how well our obtained ranking orders comply with given target ranking orders, we require a method to quantify the agreement of two ranking orders. In the following we will present some ranking measures.

7.1.1 Spearman's Footrule

Spearman's Footrule [40] is a very simple and intuitive ranking measure. It essentially counts the positional difference between the computed rank and the target rank for each element and then takes the sum of all individual displacements. Mathematically it is defined as

$$F(\overline{R}, R) = \sum_{i=1}^{n} |\overline{R}_i - R_i|, \tag{7.1}$$

where R and $\overline{R}$ are ranking order vectors such that $\overline{R}_i, R_i \in [n]$ for all nodes v_i.

Spearman's Footrule is especially popular for its simplicity. In order to use this measure for ranking evaluation with several query vectors and therefore several ranking vectors, we modified Equation (7.1) by dividing $F(\overline{R}, R)$ by n such that we obtain the average positional distance for each node, calculating this measure for each query vector and then taking the average. Our modified definition is now given as

$$F(\overline{R}, R) = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{n} \sum_{i=1}^{n} |\overline{R}_{qi} - R_{qi}|. \tag{7.2}$$

7.1.2 (Normalized) Discounted Cumulative Gain

Discounted Cumulative Gain (DCG) [41] is a ranking measure which takes the importance of every node into account. It therefore requires a relevance function $rel : [n] \to \mathbb{R}$ which assigns every ranking position an importance score. The DCG score of a computed ranking is then defined as

$$DCG(\overline{R}, R) = \sum_{i=1}^{n} \frac{rel(R_i)}{\log(\overline{R}_i + 1)}. \tag{7.3}$$

The higher the DCG score of a ranking, the better its compliance with the target ranking.

One problem with the Discounted Cumulative Gain is that its result is unbounded such that it could reach infinitely large values. It is therefore possible to compare two computed rankings with each other if they have the same target ranking, however, it does not provide an intuitive explanation of the compliance of a computed ranking with the target ranking.

In order to bound the DCG score to a certain scale it is necessary to normalize it. The so obtained ranking measure is called Normalized Discounted Cumulative Gain (NDCG) [41]. Formally, it is defined as

$$NDCG(\overline{R}, R) = \frac{DCG(\overline{R}, R)}{DCG(\overline{R}, \overline{R})}. \tag{7.4}$$

It essentially calculates the ratio of the DCG score of the computed ranking and the DCG score of a perfectly compliant ranking order.

We can furthermore modify (N)DCG in order to obtain (N)DCG@k which only considers the discounted relevance of the top-k nodes in the target ranking.

Throughout this thesis we will assume that the relevance function is linearly decreasing for each position such that the relevance of the most important node is n, for the second most important node $n-1$ and for the least important node 1.

7.1.3 Precision Measures

The accordance of a ranking with a target ranking can furthermore be quantified by precision measures. These require a parameter k and then assume that only the top-k nodes in the target ranking are relevant while all other nodes are irrelevant. The obtained ranking measure is then called "Precision@k" [42]. It calculates the fraction of relevant nodes in the k most important nodes of the computed ranking. More formally, Precision@k is defined as

$$P_k(\overline{R}, R) = \frac{1}{k} \sum_{i:R_i<=k} \mathbb{1}(\overline{R}_i <= k). \tag{7.5}$$

Another precision measure for rankings is Average Precision [41]. It again assumes that the top-k nodes of the target ranking are relevant and all other nodes are irrelevant. The Average Precision of a ranking order R is then calculated as follows. First the computed ranking positions l_i of all k relevant nodes are determined. For each l_i one then calculates the ratio of relevant and irrelevant nodes in the first l_i computed ranks. The average of these scores is then the Average Precision. Formally Average Precision is defined as

$$AP_k(\overline{R}, R) = \frac{1}{k} \sum_{i:R_i<=k} \frac{\sum_{j=1}^{i} \mathbb{1}(\overline{R}_j <= k)}{i}. \tag{7.6}$$

7.2 Scalability

In this section we will provide an analysis of the efficiency of our two order loss approaches. We will start with our conversion approach and then discuss the scalability of our Hinge loss approach.

7.2.1 Conversion to Ranking Scores

Our first approach uses a decreasing function like exponential decay to convert the given ranking order to ranking scores. Afterwards we can then use our basic approach for learning the edge weights from given ranking scores (see Chapter 3). The conversion has a time complexity of $\mathcal{O}(|Q|n)$ since all n ranking positions for each of the $|Q|$ query vectors need to be converted. Since the weight learning process is then equivalent to the value loss approach (see Chapter 3), the time complexity is also equivalent as in Section 6.1.

7.2.2 Hinge Loss

In order to evaluate the efficiency of our Hinge loss approach we conducted the same experiment as in Section 6.1. We again used two Barabási-Albert-Networks with different node counts and increased the branching factor λ to see how our algorithm depends on the number of edges. Again, we used a real-world dataset and increased the number of nodes and therefore also the number of edges to check the behaviour of our approach. The real-world dataset hereby was the Netscience network. It can be seen that scaling for the synthetic networks is again approximately linear. However, the runtime difference between the smaller and larger network is much higher than in the value loss case. While the lines in the value loss case were roughly parallel, the difference in the slopes for the order loss case is significant. This indicates that the number of nodes has a larger impact for the Hinge loss approach than for the value loss approach. This indication is affirmed by the results of the real-world network. While the runtime scaling of the real-world network test in the value loss experiment was still linear, Figure 7.1 now shows a quadratic dependence. This dependence was expected since it can be seen from Equation (4.4) that the objective is quadratic in the number of nodes.

The influence of the amount of query vectors on the total runtime are equivalent to the influence as described in Section 6.2.

We furthermore analysed the efficiency of our top-k Hinge loss approach. The top-k approach for our value loss and therefore also for our conversion order loss approach had to set the ranking scores of non-top-k nodes to zero and were consequently not able to reduce the runtime but actually increased it. However, our Hinge loss top-k method is actually more efficient than the full ranking method since it is now not necessary to iterate over all $n * (n-1)$ pairs but only over $k * (n-1)$ pairs, which is way faster since $k \ll n$ in most cases. To demonstrate the actual amount of saved time, we ran our Hinge

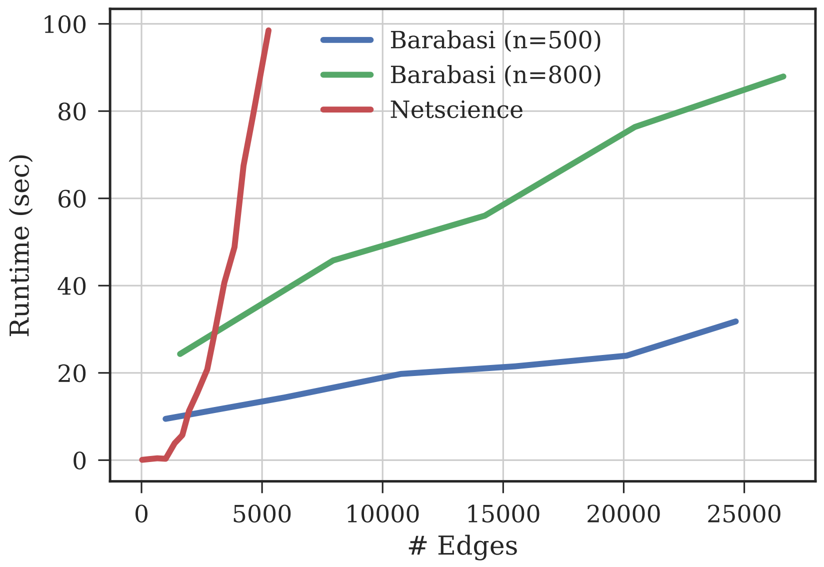

Figure 7.1: Impact of the edge count on the runtime of our algorithm. We used 100 iterations and 1 query vector to obtain these results.

	k		
	10	100	n^*
Barabási Small	**2.88**	22.6	74.34
Netscience	**1.37**	9.17	44.98
Cora	**13.68**	133.22	357.95

*: $k = n$, full ranking was used

Table 7.1: Runtime (sec) Comparison for different k in the top-k Hinge loss approach using 1 query vector and 50 iterations.

loss algorithm for multiple datasets with different values of k and compared the respective runtimes with the runtime of the full ranking variant. The result of this experiment is shown in Table 7.1 and it can be seen that the top-k approach has a huge impact on the total runtime.

7.3 Accuracy

Now that we have analysed the efficiency of our algorithm, it remains to show that we are actually able to learn weights such that our obtained ranking order corresponds approximately to the target ranking. We will again first demonstrate the convergence of our approaches and then evaluate the weight recovering quality.

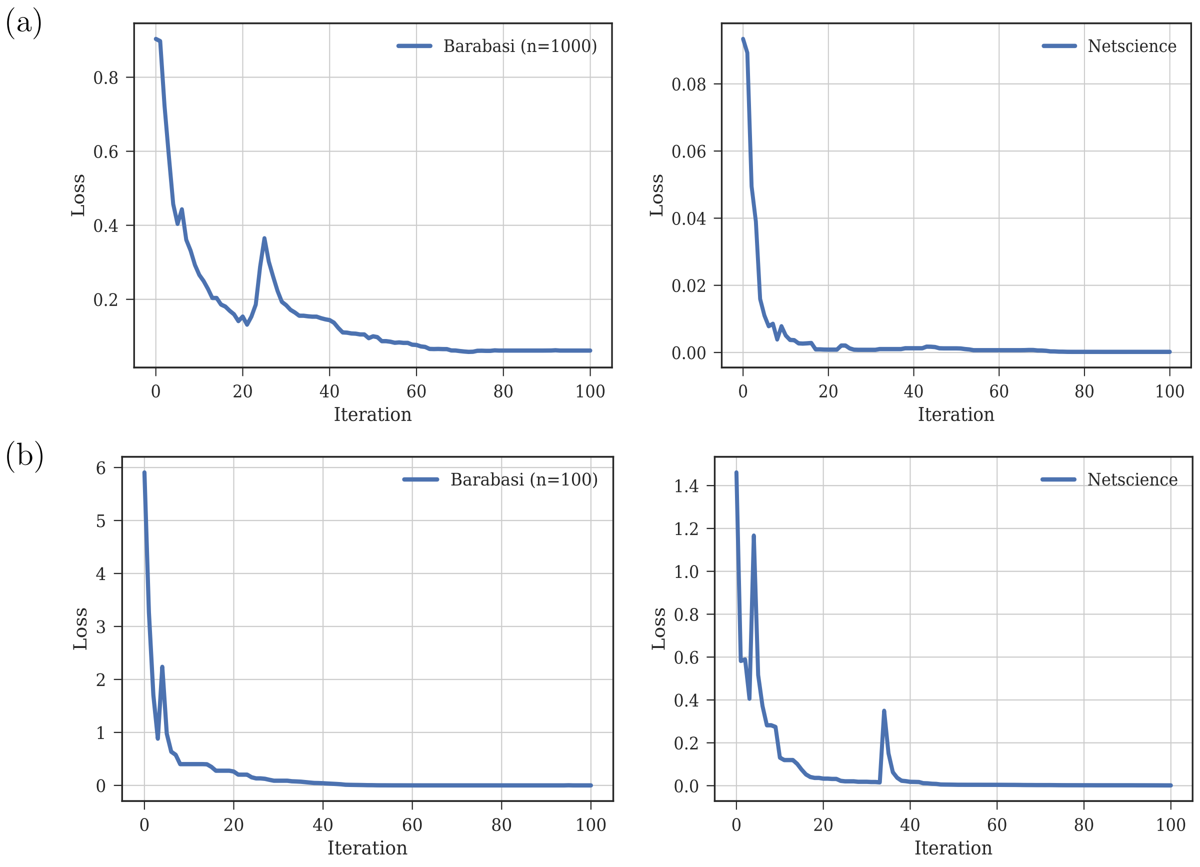

Figure 7.2: Convergence of the objective function. (a) Naïve Approach (b) Hinge Loss

7.3.1 Optimization

In order to verify the correctness of our approach we again start by checking the converge of our objective loss function for both order loss approaches. Figure 7.2 shows the trend of our objective function for 100 iterations of each approach for a synthetic and a real-world dataset. It can be seen that our algorithm indeed converges to zero.

7.3.2 Recovering Quality

Next we need to evaluate how well our approaches are able to recover the target ranking order and the original weights. We will start by analysing the accordance of our obtained order with the target order for both approaches. Hence, we ran our algorithm on several real-world networks and compared the results according to the ranking measures described in Section 7.1. We furthermore compared the results for several k values. Table 7.2 shows the result of this experiment.

	k	Sp @ 10	Sp @ 100	Sp @ n	NDCG @ 10	NDCG @ 100	NDCG @ n	Prec @ 10	Prec @ 100	RT (sec)
Barabasi Small Naïve	10	17.00	122.83	226.95	0.911	0.898	0.965	0.60	0.51	2.46
	100	48.80	136.43	199.84	0.923	0.906	0.970	0.70	0.60	5.52
	n	7.50	66.88	186.88	0.932	0.938	0.979	0.70	0.65	**1.11**
Barabasi Small Hinge	10	3.60	72.97	183.97	0.921	0.936	0.977	0.60	0.63	2.65
	100	**0.30**	23.31	159.05	**0.998**	0.982	0.986	**0.90**	0.84	16.46
	n	2.80	**21.38**	**68.56**	0.974	**0.987**	**0.997**	0.80	**0.85**	60.18
Cora Naïve	10	6.50	980.07	918.52	0.919	0.653	0.915	0.70	0.13	**3.02**
	100	122.20	202.90	869.48	0.726	0.899	0.935	0.30	0.58	8.76
	n	238.50	285.57	299.78	0.763	0.846	0.987	0.50	0.45	3.49
Cora Hinge	10	**0.00**	40.16	167.02	**1.000**	0.960	0.997	**1.00**	0.74	42.22
	100	**0.00**	**2.39**	152.90	**1.000**	**0.999**	0.998	**1.00**	**0.95**	109.42
	n	3.20	19.92	**34.17**	0.953	0.988	**0.999**	0.80	0.82	324.73
Last.fm Naïve	10	**0.00**	523.86	572.92	**1.000**	0.696	0.944	**1.00**	0.13	3.52
	100	6.80	22.54	484.19	0.962	0.988	0.965	0.80	0.84	4.23
	n	10.60	25.00	18.88	0.921	0.982	0.999	0.60	0.83	**3.10**
Last.fm Hinge	10	1.00	18.83	14.27	0.969	0.990	**1.000**	0.80	0.87	20.01
	100	0.20	4.95	12.73	**1.000**	0.996	**1.000**	**1.00**	0.91	34.30
	n	0.80	**3.21**	**10.45**	**1.000**	**1.000**	**1.000**	**1.00**	**0.97**	240.64
Netscience Naïve	10	**0.00**	**0.00**	**0.00**	**1.000**	**1.000**	**1.000**	**1.00**	**1.00**	**0.76**
	100	**0.00**	0.16	0.01	**1.000**	**1.000**	**1.000**	**1.00**	**1.00**	2.42
	n	0.20	0.02	**0.00**	**1.000**	**1.000**	**1.000**	**1.00**	**1.00**	5.85
Netscience Hinge	10	0.10	14.43	5.69	0.998	0.994	**1.000**	0.90	0.92	5.39
	100	**0.00**	**0.00**	**0.00**	**1.000**	**1.000**	**1.000**	**1.00**	**1.00**	1.55
	n	**0.00**	**0.00**	**0.00**	**1.000**	**1.000**	**1.000**	**1.00**	**1.00**	11.50

Sp: Spearman, Prec: Precision, RT: Runtime

Table 7.2: Test Results of our two order loss approaches based on several ranking measures for different top-k values. We always used 100 iterations and 10 query vectors. Best values are marked in bold.

It can be seen that our conversion approach works well for some networks, especially the Netscience dataset. However, for some networks it is unable to find a fitting ranking order. Furthermore we noticed that the Hinge loss approach is quite consistent in finding good ranking orders for all networks, it is even able to find fitting weight assignments for networks where the conversion approach achieves only unsatisfying results (e.g. Cora and Barabási). Moreover, the Hinge loss method obtains better results for all datasets. It is interesting to see that the results achieved by the top-k Hinge loss approach were not significantly worse than for the full ranking. Due to the enormous reduction of runtime by using only the top-k values, it is a reasonable alternative. However, compared both the top-k and the full ranking Hinge loss approach are compared to the conversion method computationally very demanding.

It remains to compare our different extensions to the Hinge loss approach, namely the addition of a margin or of a position-aware scaling factor. We therefore ran our Hinge loss algorithm with each of these extensions and compared the results which are displayed in Table 7.3. Unfortunately, neither the margin nor the position-awareness extension were able to improve the accuracy but only achieved similar results.

We also checked whether the extensions improved the accuracy in the top-k setting, but the obtained results were also quite similar to those of the normal Hinge loss.

Now only the question remains whether our order approaches are able to learn an edge weight assignment close to the original weights. We already observed in Chapter 6.2 that there are multiple weight assignments which result in quite similar PageRank vectors. In the order loss case, there additionally exist multiple - possibly very different - ranking score vectors which yield the same ranking order which makes it pretty unlikely to recover the original weights. Our conversion approach could probably only obtain the true weight assignment if the converted ranking score vector is by chance roughly equal to the true ranking score. The Hinge loss approach would stop once a ranking order is found which is identical to the target order. The true weights could therefore only be recovered if the first identical ranking has a ranking score vector equal to the true ranking scores. As it can be seen in Table 7.4, the recovered weights are in most cases quite different from the original weights although the ranking order is quite similar to the target order.

	Sp @ 10	Sp @ 100	Sp @ n	NDCG @ 10	NDCG @ 100	NDCG @ n	Prec @ 10	Prec @ 100
Barabasi Small Normal	4.00	22.25	**57.72**	0.970	**0.986**	**0.998**	**0.80**	0.82
Barabasi Small Margin	5.30	34.26	96.46	0.919	0.967	0.994	0.70	0.74
Barabasi Small Pos.Aware	**2.90**	**21.29**	71.50	**0.986**	0.985	0.997	**0.80**	**0.86**
Cora Normal	**1.10**	**9.37**	**30.26**	0.989	**0.996**	**1.000**	0.80	**0.93**
Cora Margin	2.10	21.79	119.95	0.960	0.980	0.998	0.80	0.79
Cora Pos.Aware	1.60	17.44	37.78	**1.000**	0.985	**1.000**	**1.00**	0.84
Last.fm Normal	**0.00**	**0.00**	**0.00**	**1.000**	**1.000**	**1.000**	**1.00**	**1.00**
Last.fm Margin	**0.00**	13.93	13.32	**1.000**	0.993	**1.000**	**1.00**	0.88
Last.fm Pos.Aware	0.30	8.97	10.81	0.998	0.995	**1.000**	0.90	0.90
Netscience Normal	**0.00**	**0.00**	**0.00**	**1.000**	**1.000**	**1.000**	**1.00**	**1.00**
Netscience Margin	**0.00**	**0.00**	**0.00**	**1.000**	**1.000**	**1.000**	**1.00**	**1.00**
Netscience Pos.Aware	**0.00**	1.28	1.80	**1.000**	**1.000**	**1.000**	**1.00**	0.98

Sp: Spearman, Prec: Precision, RT: Runtime

Table 7.3: Result Comparison for our Hinge Loss extensions: Margin and Position-Awareness. Best values are marked in bold.

	k	Mean Weight Distance (L1)	Mean Weight Distance (L1)
Barabasi Small Naïve	10	**0.134**	**2.92e-03**
	100	**0.134**	**2.92e-03**
	n	**0.134**	**2.92e-03**
Barabasi Small Hinge	10	0.133	2.91e-03
	100	0.132	2.90e-03
	n	**0.124**	**2.76e-03**
Cora Naïve	10	0.075	1.15e-03
	100	0.075	1.15e-03
	n	**0.074**	**1.13e-03**
Cora Hinge	10	0.078	1.21e-03
	100	0.075	1.15e-03
	n	**0.059**	**9.47e-04**
Last.fm Naïve	10	0.128	2.40e-03
	100	**0.127**	**2.38e-03**
	n	0.128	2.39e-03
Last.fm Hinge	10	**0.128**	**2.38e-03**
	100	0.167	2.94e-03
	n	0.167	2.91e-03
Netscience Naïve	10	**0.028**	**8.85e-04**
	100	0.029	8.95e-04
	n	0.029	8.86e-04
Netscience Hinge	10	**0.030**	**8.82e-04**
	100	0.104	2.02e-03
	n	0.104	2.00e-03

Table 7.4: Weight Recovery Results for our both order loss approaches. 100 iterations and 1 query vector were used each time. Best values are marked in bold.

7.4 Conclusion

In this chapter we analysed the efficiency and effectiveness of our two approaches to learn edge weights such that the obtained ranking order corresponds to a given target order. Our first approach converts the ranking order into ranking scores and then uses our value loss method (see Chapter 3) in order to learn a weight assignment while our other approach uses a pair-wise Hinge loss to obtain an order close to the target order.

While our first approach is quite efficient, our Hinge loss is computationally very demanding due to the necessity to iterate over all possible node pairs. However, we can reduce the time complexity of the Hinge loss method by only considering the loss for the top-k nodes. Nevertheless, Hinge loss is still quite expensive compared to the conversion approach.

Despite the higher time complexity, we still think that Hinge loss is the better way to learn weights from a target order since it is more accurate and more consistent over several datasets than the conversion approach.

Chapter 8

Application: Spectral Clustering

After covering the theoretical details and the experimental evaluation of Network Learning it remains to be seen whether it can improve the practical performance of graph algorithms. This chapter will now answer this question for the case of spectral clustering on a network. We first describe how our experiments were set up and then present numerical results showing the impact of weights on the clustering result. Finally, we will draw a conclusion whether network learning is a useful technique to enhance the performance of machine learning algorithms.

8.1 Setup

We used two different datasets to evaluate the clustering performance on the original weighted network, the unweighted network, a network with uniform weights and one network each for value loss and order loss learned weights.

Our first dataset is Cora. As mentioned in Section 5.1, nodes in this network represent scientific publications, and edges represent citations. Furthermore, each publication belongs to one of seven categories. We can therefore use these categories as ground truth to evaluate the accuracy of spectral clustering. The second network is "Primary School Contacts". In this case the ground truth clusters are the school classes each child belongs to.

We then learned edge weights for each network with two of our approaches, first the value loss method as described in Section 3 and the Hinge loss approach (see Section 4) using either 1, 10 or 100 query vectors.

8.2 Results

We then ran a spectral clustering algorithm on the networks described above and compared the obtained clusterings to the ground truth clustering by quantifying the accordance of our obtained clustering results with the ground truth clustering using two cluster evaluation measures. First the *Fowlkes-Mallows-Index* (FMI) [43, 44] which is defined as

$$FMI = \sqrt{\frac{TP}{TP+FP} \cdot \frac{TP}{TP+FN}}, \tag{8.1}$$

where TP stands for *True Positives*, FP for *False Positives* and FN for *False Negatives.* True positive in this context denotes the number of point-pairs which are in the same cluster for both the ground truth and the obtained clustering. False positives are point-pairs that do not belong to the same ground truth cluster but were clustered together. Lastly, false negatives stands for the point-pairs which should be in the same cluster but are not. FMI returns a scalar between zero and one, where higher values correspond to a better clustering result.

The second measure we used for evaluation is *Normalized-Mutual-Information* (NMI) which quantifies the shared information between two clusterings $\mathcal{C}_1, \mathcal{C}_2$ [45]. Normalized-Mutual-Information is formally defined as:

$$NMI(\mathcal{C}_1, \mathcal{C}_2) = \frac{I(\mathcal{C}_1, \mathcal{C}_2)}{\sqrt{H(\mathcal{C}_1)H(\mathcal{C}_2)}}, \tag{8.2}$$

where $I(X, Y)$ denotes the mutual information score as defined in [46] and $H(\mathcal{C}_1)$ the entropy of $\mathcal{C}_1$. As for FMI, the best possible value for NMI is 1.0 and the worst is 0.0.

The results for the Cora network can be seen in Table 8.1. It can be seen that in general no accurate clustering results could be achieved. A possible reason for this might be that the cosine similarity we used as edge weights (see Section 5.1) does not accurately describe the actual similarity between two publications. We can also see that Cora can actually be better clustered using an unweighted network or a network with uniform weights which is another indication that the edge weights are not corresponding to the actual similarity. The value loss approach achieves quite similar results as the ones obtained using the original weighted network. Surprisingly, the order loss approach achieves by far the best results. A possible explanation for this anomaly is that the order loss approach could find

	$\|Q\|$	Normalized Mutual Information	Fowlkes Mallows Index
Original Weights	-	0.187	0.335
Uniform Weights	-	0.241	0.406
Unweighted	-	0.202	0.377
Value Loss	1	0.162 ± 0.000	0.361 ± 0.010
	10	0.182 ± 0.039	0.380 ± 0.010
	100	0.183 ± 0.044	0.382 ± 0.009
Order Loss	1	0.252 ± 0.005	0.412 ± 0.002
	10	**0.324** ± 0.055	0.435 ± 0.016
	100	0.301 ± 0.047	**0.438** ± 0.018

Table 8.1: Clustering result evaluation for Cora. Values are averaged over 10 runs with 250 iterations of our algorithm. ± denotes the standard deviation. Best values are marked in bold.

	$\|Q\|$	Normalized Mutual Information	Fowlkes Mallows Index
Original Weights	-	**0.992**	**0.991**
Uniform Weights	-	0.951	0.924
Unweighted	-	0.937	0.902
Value Loss	1	0.959 ± 0.007	0.925 ± 0.011
	10	0.955 ± 0.015	0.903 ± 0.038
	100	0.97 ± 0.018	0.935 ± 0.046
Order Loss	1	0.964 ± 0.004	0.941 ± 0.009
	10	0.954 ± 0.052	0.928 ± 0.091
	100	0.991 ± 0.003	**0.991** ± 0.003

Table 8.2: Clustering result evaluation for the Primary School Contact dataset. Values are averaged over 10 runs with 250 iterations of our algorithm. ± denotes the standard deviation. Best values are marked in bold.

a weight assignment dissimilar from the original weights which achieves about the same ranking order but can be better clustered.

Table 8.2 shows the results obtained for the Primary School Contacts network. Here we can actually see a benefit of weights on the performance of spectral clustering since the results on the original weighted network achieved better scores than clustering on the unweighted network. We can furthermore see that our weight learning methods achieve better results than the unweighted network in all cases except for the value loss approach with 10 query vectors measured using FMI.

8.3 Conclusion

Unfortunately, we could not observe a general clustering improvement when using learned weights instead of uniform weights or no weights. For the Cora network, uniform weights achieved even better results than the original weights. However, for the Primary School Contacts network we could observe that our learned weights actually could improve the clustering performance compared to the uniformly weighted and unweighted network.

Chapter 9

Application: Link Prediction

After analysing the impact of weights on the performance of spectral clustering in the previous chapter we now want to do the same for link prediction. We again first describe our experimental setup and then present the results and finally draw a conclusion whether our weight learning techniques are able to improve the performance of link prediction.

9.1 Setup

For our experiments we used the DBLP datasets and the Irvine communication dataset (see Section 5.1). As described in Section 2.3 we first need to split the datasets into a training network G_{train} and a test network G_{test}. The DBLP sets were split such that the training network contained all collaborations in the time interval $[2001, 2005]$ and the test network all collaborations in $[2006, 2010]$. The Irvine network was split such that the training network contained all connections before or on 31st May 2004 and the test network all connections after that date. We then chose to remove all nodes with less than 2 adjacent nodes in either the training or the test network. Table 9.1 shows the node and edge counts for all obtained networks.

In the next step we learned edge weights for all obtained networks using first our value loss approach and second our Hinge order loss approach using 250 iterations and 25 randomly chosen query vectors.

	G_{train}		G_{test}	
	$\lvert V\rvert$	$\lvert E_{old}\rvert$	$\lvert V\rvert$	$\lvert E_{new}\rvert$
DBLP (DB)	264	682	264	527
DBLP (IR)	145	218	145	210
DBLP (ML)	113	120	113	93
DBLP (DM)	180	294	180	209
Irvine	806	5877	806	2355

Table 9.1: Node and Edge Counts for training network G_{train} and test network G_{test} for all Datasets used for Link Prediction. The edges are undirected in this case.

9.2 Results

In this section we will compare link prediction results for all datasets described above. In particular we will perform link prediction using weighted, unweighted, value-loss-learned and order-loss-learned networks. In order to predict links we use the three similarity score measures Katz Measure, Personalized PageRank and Negated Shortest Paths as described in Section 2.3. We then pick the $n = |E_{new}|$ node pairs with the highest similarity scores and calculate the precision in order to compare the performance for the different networks. Table 9.2 shows our obtained results. It can be seen that for the DBLP datasets the results achieved using the original weighted network are often but not always better than the results for the unweighted network. We can also see that our learned weights in most cases where the weighted network produced better prediction results than the unweighted network also outperformed the unweighted network. However for the Irvine dataset, the link predictions based on the unweighted network achieved the best accuracy. The accuracy for the networks with learned weights for both the value loss and the order loss correspond to the accuracy obtained using the original network. In general, there is no visible trend whether the accuracy of the weighted or unweighted network is better.

9.3 Conclusion

As for spectral clustering, we were unfortunately unable to observe significant improvement of link prediction when using a weighted instead of an unweighted network. We could see that link prediction for the DBLP networks usually performed better if edge weights, either the original or the learned weights, were used. However, for the Irvine communication network it was the other way around such that our experiments were indecisive.

		Accuracy		
		Katz	Pers. PageRank	Shortest Paths
DBLP (DB)	Original Weights	**0.083**	0.093	0.068
	Unweighted	0.070	**0.095**	0.098
	Value Loss	0.063 ± 0.001	0.085 ± 0.001	0.098 ± 0.001
	Order Loss	0.066 ± 0.001	0.077 ± 0.003	**0.102** ± 0.002
DBLP (IR)	Original Weights	**0.271**	**0.290**	0.355
	Unweighted	0.243	0.243	0.336
	Value Loss	0.252 ± 0.003	0.262 ± 0.001	0.346 ± 0.002
	Order Loss	0.255 ± 0.004	0.252 ± 0.007	**0.364** ± 0.000
DBLP (ML)	Original Weights	**0.162**	**0.162**	**0.108**
	Unweighted	0.108	**0.162**	**0.108**
	Value Loss	**0.162** ± 0.000	**0.162** ± 0.000	**0.108** ± 0.000
	Order Loss	**0.162** ± 0.000	**0.162** ± 0.000	0.081 ± 0.000
DBLP (DM)	Original Weights	0.109	0.139	0.089
	Unweighted	**0.129**	0.129	0.079
	Value Loss	0.089 ± 0.004	**0.158** ± 0.003	0.079 ± 0.005
	Order Loss	0.106 ± 0.004	0.145 ± 0.005	**0.119** ± 0.008
Irvine	Original Weights	0.018	0.033	0.024
	Unweighted	**0.021**	**0.040**	**0.046**
	Value Loss	0.017 ± 0.001	0.032 ± 0.004	0.036 ± 0.002
	Order Loss	0.020 ± 0.001	0.036 ± 0.000	0.041 ± 0.002

Table 9.2: Link Prediction Precision for several datasets obtained by using either the original weighted network, an unweighted network or a network with learned edge weights either using a value loss approach or an order loss approach. The results for the learned networks were averaged over 10 runs. ± denotes the standard deviation. Best values marked in bold.

Chapter 10

Conclusion and Future Work

In this thesis we presented several approaches to learn edge weights of a network for a given target ranking. Our first approach uses given PageRank scores to find an optimal weight assignment by minimizing a $\ell 2$-Loss between the target ranking and a computed ranking. We determined through various experiments that our algorithm is able to recover the original weights very well although in some cases a different weight assignment with almost identical ranking scores was found.

Furthermore, we proposed two methods which only require a ranking order of the network nodes instead of numerical ranking scores. The first of these approaches learns edge weights by first converting the target order back to numerical ranking values such that our first approach can be used while our second order loss method uses a Hinge loss between the computed ranking scores if the two nodes are not in the same order as in the target order. Our experimental evaluation of these methods determined that the Hinge loss approach achieves more accurate results but comes with the drawback of high computational expense.

Moreover, we proposed solutions for both the value loss and the order loss scenario where not a full ranking but only a partial ranking consisting of the top-k nodes was given. We showed in our evaluations that partial rankings do not significantly reduce the accuracy of our algorithms.

Besides developing algorithmic approaches for network learning we also tested their impact on the performance of two Machine Learning algorithms, namely spectral clustering and link prediction. Unfortunately, we could not determine a beneficial influence of edge weights compared to the performance of these algorithms on unweighted or uniformly weighted networks.

The performance of spectral clustering and link prediction apparently mostly depends on the graph structure and less on the actual weights. It might therefore be interesting for future work to determine whether it is possible to learn the graph structure from a given ranking and not only a weight assignment.

List of Figures

List of Tables

Bibliography

[1] Aidong Zhang. *Protein Interaction Networks: Computational Analysis.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[2] Peter Bearman, James Moody, and Robert Faris. Networks and history. *Complex.*, 8(1):61–71, September 2002.

[3] P. Latouche and F. Rossi. Graphs in machine learning: an introduction. *ArXiv e-prints*, June 2015.

[4] Stephan M. Wagner and Nikrouz Neshat. Assessing the vulnerability of supply chains using graph theory. *International Journal of Production Economics*, 126(1):121 – 129, 2010. Improving Disaster Supply Chain Management – Key supply chain factors for humanitarian relief.

[5] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[6] Chin-Chi Hsu, Yi-An Lai, Wen-Hao Chen, Ming-Han Feng, and Shou-De Lin. Unsupervised ranking using graph structures and node attributes. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 771–779, New York, NY, USA, 2017. ACM.

[7] W. Xing and A. Ghorbani. Weighted pagerank algorithm. In *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*, pages 305–314, May 2004.

[8] Rebecca S. Wills. Google's pagerank: The math behind the search engine. *Math. Intelligencer*, pages 6–10, 2006.

[9] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, March 1953.

[10] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information*

and Knowledge Management, CIKM '03, pages 556–559, New York, NY, USA, 2003. ACM.

[11] Jon M. Kleinberg. Hubs, authorities, and communities. *ACM Comput. Surv.*, 31(4es), December 1999.

[12] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 849–856, Cambridge, MA, USA, 2001. MIT Press.

[13] U. von Luxburg. A Tutorial on Spectral Clustering. *ArXiv e-prints*, November 2007.

[14] Venu Satuluri and Srinivasan Parthasarathy. Symmetrizations for clustering directed graphs. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 343–354, New York, NY, USA, 2011. ACM.

[15] F. D. Malliaros and M. Vazirgiannis. Clustering and community detection in directed networks: A survey. 533:95–142, December 2013.

[16] Panagiotis Symeonidis and Christos Perentis. Link prediction in multi-modal social networks. In *Proceedings of the 2014th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III*, ECMLPKDD'14, pages 147–162, Berlin, Heidelberg, 2014. Springer-Verlag.

[17] Nitin Chiluka, Nazareno Andrade, and Johan Pouwelse. *A Link Prediction Approach to Recommendations in Large-Scale User-Generated Content Systems*, pages 189–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[18] Chengwei Lei and Jianhua Ruan. A novel link prediction algorithm for reconstructing protein–protein interaction networks by topological similarity. *Bioinformatics*, 29(3):355–364, February 2013.

[19] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Comput. Surv.*, 49(4):69:1–69:33, December 2016.

[20] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 271–279, New York, NY, USA, 2003. ACM.

[21] Bryan Sims, Harish Bhat, Roummel Marcia, Avi Shapiro, and Boaz Ilan. Investorrank and an inverse problem for pagerank. 2012.

[22] Romain Hollanders, Jean-Charles Delvenne, and Raphaël M. Jungers. Policy iteration is well suited to optimize pagerank. *CoRR*, abs/1108.3779, 2011.

[23] Cristobald de Kerchove, Laure Ninove, and Paul Van Dooren. Maximizing pagerank via outlinks. *CoRR*, abs/0711.2867, 2007.

[24] A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, October 1999.

[25] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[26] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[27] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. 74(3):036104, September 2006.

[28] M. E. J. Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical Review E*, 64(1):016132+, June 2001.

[29] Advogato network dataset – KONECT, April 2017.

[30] Paolo Massa, Martino Salvetti, and Danilo Tomasoni. Bowling alone and trust decline in social network sites. In *Proc. Int. Conf. Dependable, Autonomic and Secure Computing*, pages 658–663, 2009.

[31] Valerio Gemmetto, Alain Barrat, and Ciro Cattuto. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC infectious diseases*, 14(1):695, December 2014.

[32] Juliette Stehlé, Nicolas Voirin, Alain Barrat, Ciro Cattuto, Lorenzo Isella, Jean-François Pinton, Marco Quaggiotto, Wouter Van den Broeck, Corinne Régis, Bruno Lina, and Philippe Vanhems. High-resolution measurements of face-to-face contact patterns in a primary school. *PLOS ONE*, 6(8):e23176, 08 2011.

[33] Uc irvine messages network dataset – KONECT, April 2017.

[34] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. *Social Networks*, 31(2):155–163, 2009.

[35] Dblp network dataset – KONECT, April 2017.

[36] Michael Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *Proc. Int. Symposium on String Processing and Information Retrieval*, pages 1–10, 2002.

[37] Jingchao Ni, Hanghang Tong, Wei Fan, and Xiang Zhang. Inside the atoms: Ranking on a network of networks. In *Proceedings of the 20th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, KDD '14, pages 1356–1365, New York, NY, USA, 2014. ACM.

[38] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, May 2006.

[39] HSL. A collection of fortran codes for large scale scientific computation. `http://www.hsl.rl.ac.uk/`.

[40] Ravi Kumar and Sergei Vassilvitskii. Generalized distances between rankings. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 571–580, New York, NY, USA, 2010. ACM.

[41] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhiming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. In *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, NIPS'09, pages 315–323, USA, 2009. Curran Associates Inc.

[42] Yi Fang and Mengwen Liu. A unified energy-based framework for learning to rank. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 171–180, New York, NY, USA, 2016. ACM.

[43] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.

[44] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2-3):107–145, December 2001.

[45] Alexander Strehl and Joydeep Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, March 2003.

[46] Amir Alush, Avishay Friedman, and Jacob Goldberger. Pairwise clustering based on the mutual-information criterion. *Neurocomput.*, 182(C):284–293, March 2016.

Appendix A

PageRank Differentiation

Implicit Differentiation of the PageRank equation (2.5):

$$\begin{aligned}
r(w) &= (1-c)(I - cA(w)^T)^{-1}q \\
(I - cA(w)^T)r(w) &= (1-c)q \\
\nabla_w((I - cA(w)^T)r(w)) &= \nabla_w((1-c)q) \\
\nabla_w(r(w) - cA(w)^T r(w)) &= 0 \\
\nabla_w(r(w)) - \nabla_w(cA(w)^T r(w)) &= 0 \\
\nabla_w(r(w)) - c\nabla_w(A(w)^T)r(w) - cA(w)^T\nabla_w(r(w)) &= 0 \\
(I - cA(w)^T)\nabla_w(r(w)) - c\nabla_w(A(w)^T)r(w) &= 0 \\
\nabla_w(r(w)) &= c(I - cA(w)^T)^{-1}\nabla_w(A(w)^T)r(w)
\end{aligned}$$

Appendix B

Dataset Statistics

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	Avg. Degree	Max,Min In-Degree	Max,Min Out-Degree	Transitivity	Avg. Clustering Coefficient
Barabási Small	1000	3992	3.99	(2, 97)	(2, 97)	1.11%	0.89%
Barabási Large	5000	49950	9.99	(5, 266)	(5, 266)	0.85%	0.10%
Cora	2485	10138	4.08	(1, 168)	(1, 168)	9.00%	5.40%
Last.fm (Subset)	3091	7318	2.37	(0, 55)	(1, 13)	20.46%	4.09%
Netscience	1461	5484	3.75	(1, 34)	(1, 34)	69.34%	22.03%
Advogato	3830	42519	11.10	(0, 721)	(1, 692)	7.63%	3.76%
Primary School Contact	232	15712	67.72	(19, 130)	(19, 130)	48.60%	0.98%

Statistics about the Irvine Communication networks and the DBLP collaboration networks are not included here because they had to be split into two networks each and had to be heavily pre-processed in order to use them for link prediction (see Chapter 9).