UNIVERSITÉ DE LIÈGE
Faculté des Sciences Appliquées
Institut d'Électricité Montefiore
RUN - Research Unit in Networking

Learning to Predict End-to-End Network Performance

Yongjun Liao

Thèse présentée en vue de
l'obtention du titre de Docteur
en Sciences de l'Ingénieur

Année Académique 2012-2013

Abstract

The knowledge of end-to-end network performance is essential to many Internet applications and systems including traffic engineering, content distribution networks, overlay routing, application-level multicast, and peer-to-peer applications. On the one hand, such knowledge allows service providers to adjust their services according to the dynamic network conditions. On the other hand, as many systems are flexible in choosing their communication paths and targets, knowing network performance enables to optimize services by e.g. intelligent path selection.

In the networking field, end-to-end network performance refers to some property of a network path measured by various metrics such as round-trip time (RTT), available bandwidth (ABW) and packet loss rate (PLR). While much progress has been made in network measurement, a main challenge in the acquisition of network performance on large-scale networks is the quadratical growth of the measurement overheads with respect to the number of network nodes, which renders the active probing of all paths infeasible. Thus, a natural idea is to measure a small set of paths and then predict the others where there are no direct measurements. This understanding has motivated numerous research on approaches to network performance prediction.

Commonly, the success of a prediction system is built on its scalability, efficiency, accuracy and practicability. For network performance prediction, two specific requirements have to be met. First, the prediction system should have a decentralized architecture which allows the natural deployment of the system within a networked application. Second, as different performance metrics are useful for different applications, the prediction system should be general and flexible to deal with various metrics in a unified framework.

This thesis presents practical approaches to network performance prediction. There are three main contributions. First, the problem of network performance prediction is formulated as a matrix completion problem where the matrix contains performance measures between network nodes with some of them known and the others unknown and thus to be filled. This new formulation is advantageous in that it is flexible to deal with various metrics in a unified framework, despite their diverse nature. The only requirement is that the matrix to be completed has a low-rank characteristic, which has long been observed in performance matrices constructed from various networks and in various metrics.

Second, the matrix completion problem is solved by a novel approach called *Decentralized Matrix Factorization by Stochastic Gradient Descent* (*DMFSGD*). The approach requires neither explicit constructions of matrices nor special nodes such as landmarks and central servers. Instead, by letting network nodes exchange messages with each other, matrix factorization is collaboratively and iteratively achieved at all nodes, with each node equally retrieving a number of measurements. The approach is practical in that it is simple, with no infrastructure, and is computationally lightweight, containing only vector operations.

Third, instead of the conventional representation of exact metric values, this thesis also investigates coarse performance representations including *binary classes* (The performance is classified into binary classes of either “good” or “bad”.) and *ordinal ratings* (The performance is quantized from 1 star to 5 stars.). Such more qualitative than quantitative measures not only fulfill the requirements of many Internet applications, but also reduce the measurement cost and enable a unified treatment of various metrics. In addition, as both class and rating measures

can be nicely integrated in the matrix completion framework, the same DMFSGD approach is applicable for their prediction, with little modification required.

The resulting prediction system has been extensively evaluated on various publicly-available datasets of two kinds of metrics, namely RTT and ABW. These experiments demonstrate not only the scalability and the accuracy of the DMFSGD approach but also its usability in real Internet applications. In addition, the benefits of predicting performance classes and ratings, rather than their actual values, are demonstrated by a case study on peer selection, a function that is commonly required in a number of network applications.

Acknowledgements

My Ph.D study in the past four years has been a challenging journey which will be remembered forever. I owe my sincere gratitude to many people for help of various kinds.

I have been very fortunate to have professor Guy Leduc as my supervisor. During the years when I pursued my Ph.D in the group of Research Unit of Networking (RUN) at University of Liège, Guy has continuously provided the inspiration, the encouragement and the advices that I needed. I am very grateful for and will benefit from the many things he taught me. One thing worth mentioning is that from him, I learnt that the quality of the research is much more important than the number of publications, a well-known but easily forgotten lesson which saved me a lot of time.

I benefited greatly from working with professor Pierre Geurts, who has acted as my second advisor. I have received valuable advices and guidance from him on machine learning techniques, without which, for example, I would not have got the best-paper award for my first paper.

I would like to thank my husband, Wei Du, who is always my constant source of strength. I started this PhD study because of his encouragement. He is himself a scientific researcher and constantly shares with me his research experiences and knowledge. Over the past years, we have shared a lot in our lives – passion, love, happiness, worries and even pains.

The Montefiore institute and the RUN group at University of Liège are places that are dear to me. I would like to thank all the current and former group members, including professor Laurent Mathy, professor Benoit Donnet, Dr. Sylvain Martin, Cyril Soldani, Dr. Mohamed Ali Kaafar, Dr. Bamba Gueye and Dr. François Cantin, for making a vibrant and friendly research environment.

This thesis is dedicated to my parents in China for their unconditional love and support. Special thanks are given to my sister who was brave to carry all the duties, some of which should have been mine, when my father was found to have cancer.

I also dedicate this thesis to my lovely daughter, Biyi, who has been constantly bringing so much joy and happiness in my life. Watching her growing is the biggest fun and achievement for me and for my husband.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The past decade has witnessed the rapid development and tremendous growth in data communication and information sharing via the Internet. The evolution is evidently speeding up, as the recent trend is towards single users owning multiple devices. A significant impact is that the classic client-server architecture is no more suitable for modern web services, because it often suffers from congestions and instability due to the large and bursty demands on their services.

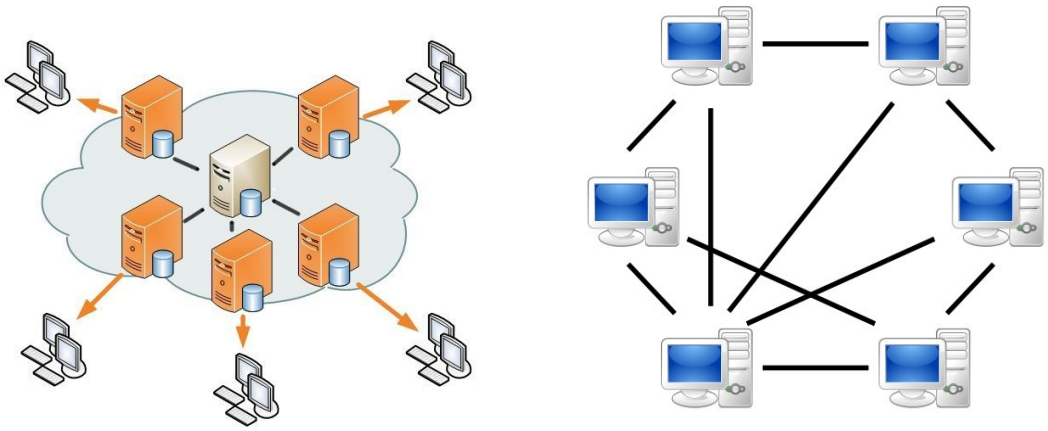
To address this issue, alternative decentralized architectures for various web services have been proposed and studied intensively. Two examples are Content Delivery Network (CDN) [11, 65, 3, 26, 72, 78] and Peer-to-Peer (P2P) Overlay Network [49, 90, 76, 54, 69].

- **Content Delivery Networks** distribute content to a set of mirror servers, scattered over the globe, so that end users can be served with the content from a nearby server in a timely and reliable manner, illustrated in Figure 1.1(a).
- **Peer-to-Peer Overlay Networks** are distributed systems with peers virtually overlayed on the IP networks, illustrated in Figure 1.1(b). In P2P overlay networks, each peer plays both roles of a client and a server, by providing services to other peers while receiving services from them.

These systems go beyond the client-server architecture in that they provide better Quality of Service (QoS) guarantees by utilizing the network resources more efficiently.

In modern web services based e.g. on CDNs and P2P networks, the knowledge of **end-to-end network performance** is desirable and beneficial. In the networking field, end-to-end network performance refers to the performance of the network path between two end nodes. Various metrics have been used to characterize the performance of network paths. For example, delay-related metrics measure the response time between network nodes and are interesting for downloading services, whereas bandwidth-related metrics indicate the data transmission rate over network paths and are of concern for online streaming.

The knowledge of end-to-end network performance can be exploited for a number of tasks that help improve modern web services, some of which are listed below.



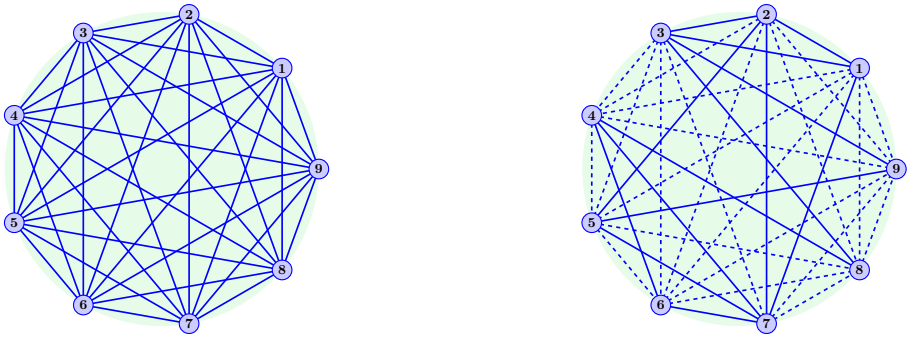
(a) content delivery networks (b) peer-to-peer overlay networks

Figure 1.1: Modern web services based on content delivery networks and peer-to-peer overlay networks.

- **Peer Selection**: In CDNs and P2P networks, each node would like to communicate with servers and peers that are likely to respond fast and well, i.e., with a connecting path of small delay or high available bandwidth. In doing so, service providers can improve accessibility and reduce the risk of having congestions [71, 66, 76, 54, 50].
- **Content Replica Placement**: In a CDN architecture, the content is replicated from the origin server to multiple mirror servers which are often at the edge of the Internet and are thus closer to end users. Performance-aware replica placement enables to optimize the distributions of the content over different mirror servers so that the server load is more balanced and the bandwidth usage is reduced [63, 17, 81, 85].
- **Overlay Construction**: In a P2P architecture, the overlay is often created by connecting nodes either in a structured manner or randomly [49]. It has been shown that the node proximity based on delay metrics can be exploited for the overlay construction, which enforces the relatively dense connections between nodes that are close to each other [55, 66, 15, 61]. Such overlays exploit better the locality principle, which states that network traffic with only local relevance should stay local [23].

Motivated by the great interests and large demands by applications, a rich body of research has been carried out on the efficient acquisition of end-to-end network performance. Generally, three major challenges are faced in the acquisition.

- **Metric Diversity:** Various metrics exist and differ largely in their characteristics and measurement methodologies. The wide diversity among these metrics renders their processing difficult in applications.
- **Costly Measurement:** Network measurement has been a fundamental problem in the history of networking. Although having been studied for decades, the measurement for some metrics still suffers from high costs and low accuracies.



(a) full mesh active measurement (b) measure a few and predict many

Figure 1.2: In (a), full mesh active measurement has a complexity of $O(n^2)$. In (b), the dashed paths are unmeasured and their performance are to be predicted from a few measurements on the solid paths. Clearly, the framework of "measure a few and predict many" is more scalable due to the reduction of measurement overheads.

- **Scalability:** It is critical to efficiently monitor the performance of the entire network. As the number of network paths grows quadratically with respect to the number of network nodes, active probing of all paths on large networks is clearly infeasible.

To solve these issues, great efforts have been put on two parallel lines of research. The first one is the study of new measurement techniques that are either more lightweight or more accurate. The other is the development of prediction schemes which allow to measure a small number of paths and then predict the performance of other paths where there are no direct measurements, as shown in Figure 1.2. Although less accurate compared to exhaustive active probing of all paths, this "measure a few and predict many" framework is much more scalable due to the significant reduction of measurement overheads. For this reason, the acquisition of network performance by prediction has attracted intensive attention in the networking community.

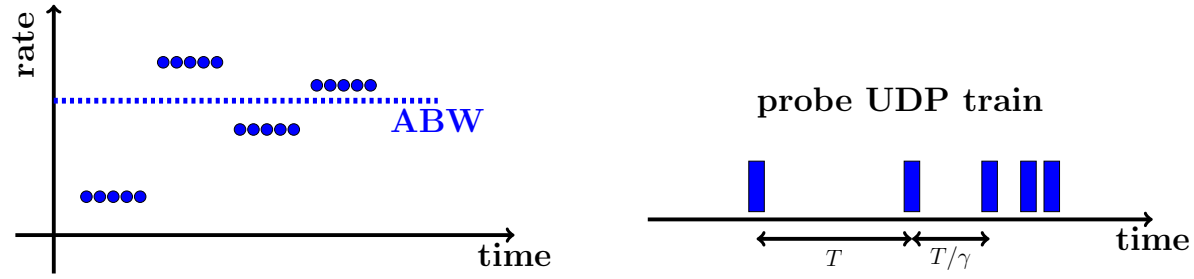
Under this background and motivated by these insights, the focus of this thesis is the accurate prediction of end-to-end network performance on large-scale networks.

1.2 Metrics of Network Performance

End-to-end network performance is a key concept at the heart of networking [21]. Popular QoS related performance metrics include round-trip time (RTT), available bandwidth (ABW) and packet loss rate (PLR). In this thesis, due to the data availability and to the relatively rare occurrence of packets losses in the Internet, only the prediction of RTT and ABW are studied.

1.2.1 Round-Trip Time (RTT)

Round-Trip Time (RTT) is one of the earliest network measurements. It is the time required for a data packet to travel from a specific source to a specific target and back again. Although the path from the source to the target is not necessarily the same as the reverse path, due to e.g. the



(a) Pathload (b) Pathchirp

Figure 1.3: Two popular ABW measurement tools. In (a), Pathload sends UDP trains (blue dots) at a constant rate, and increases the probing rate if no congestion is observed and decreases otherwise, until convergence to the ABW (dashed line). In (b), Pathchirp varies the probing rate within a train (blue rectangles) exponentially, i.e., the packets in a train are spaced exponentially.

interdomain routing policies and load balancing strategies, the RTTs between two network nodes can approximately be treated as symmetric [9, 60].

The RTT of a path can easily be obtained by using PING which sends ICMP ECHO packets to a target and captures the ECHO REPLY packets. To measure RTTs, only the senders need to be under experimental control and the target nodes respond under the ICMP protocol in a normal configuration.

1.2.2 Available Bandwidth (ABW)

Each link in a network has a physical capacity which is the maximal rate at which data can be transfered on that link. The utilization of a link is the used fraction of the link capacity, and the unused capacity is the available bandwidth (ABW).

The end-to-end ABW is the maximum data transfer rate that can be added between a sender and a receiver without congesting the path in-between. Among the links on a network path, the one with the smallest unused capacity is the bottleneck link, and the end-to-end ABW equals the ABW of the bottleneck link. In this thesis, unless stated otherwise, ABW refers to the end-to-end ABW or the ABW of a network path.

Due to the great interests by applications, various tools have been developed to measure the ABW of a path, generally based on the principle of self-induced congestion. The idea is that if the probing rate exceeds the available bandwidth over the path, then the probe packets become queued at some router, resulting in an increased transfer time. The ABW can then be estimated as the minimum probing rate that creates congestions or queuing delays. Based on this principle, Pathload [34] and Pathchirp [68] were developed, which were shown to be generally more accurate than other competitors [74].

- Pathload sends trains of UDP packets at a constant rate and adjusts the rate from train to train, according to whether congestions are observed, illustrated in Figure 1.3(a).
- Pathchirp varies the probing rate within a train exponentially, illustrated in Figure 1.3(b).

Compared to RTT, measuring ABW is much more costly and less accurate, suffering from an underestimation bias due to the bursts of network traffic and the presence of multiple links with roughly the same ABW [35]. This bias can be relieved by increasing the probe packet train length at the cost of more measurement overhead [21]. In contrast to the RTT which is inferred by the sender, the ABW is clearly asymmetric and its measurement is inferred at the target node.

1.3 Learning to Predict End-to-End Network Performance

1.3.1 Problem Statement

Formally, the problem of end-to-end network performance prediction is stated as follows. Consider a network of n end nodes. Define a path to be a routing path between a pair of end nodes, which consists of IP links between routers. There are $O(n^2)$ paths among the n end nodes, and we wish to monitor a small subset of paths so that the performance of all other paths can be estimated. In this thesis, this prediction problem is cast as a statistical inference problem, and solved by machine learning techniques.

Machine learning is a scientific discipline concerned with the design and development of algorithms that automatically learn to recognize complex patterns and make intelligent decisions based on data [6]. In using machine learning, there are two open questions to be answered.

- Which learning model is suitable for network performance prediction?
- Which and how many paths have to be monitored?

This thesis shows that the model of matrix factorization can produce accurate results when a few randomly selected paths, as few as $1 \sim 2\%$ for a network of a few thousand nodes, are monitored. Such model is founded on recent advances in machine learning and mathematics, namely Matrix Completion. In addition, its suitability for network performance prediction has a solid root in the nature of the current Internet, i.e., the usage, topology and routing.

1.3.2 System Requirements

While network performance prediction can be solved within a general learning framework, standard learning algorithms are not directly applicable to the networking environment. In the design of a practical system for supporting modern web services and emerging Internet applications, the following requirements have to be met.

- **Good Accuracies with Limited Measurements:** practical systems should achieve good accuracies with as little as possible measurement overhead, because the measurement cost may outweigh the benefits of exploiting the performance information.
- **Decentralized Architecture:** Most Internet services and applications are in nature distributed, where decentralized processing of data is most of the time a necessity. In addition, the prediction system should be scalable, lightweight and easily deployable.

- **Little Experimental Control:** For security and privacy reasons, end users and network administrators may not want to reveal information such as the IP addresses and the routing table of the network. Thus, the prediction system should be able to function with little experimental control over network nodes or routers.
- **Robustness to Network Dynamics:** The Internet environment is highly dynamic, with the performance of network paths varying over time and nodes joining and leaving the network frequently. The prediction system should be able to quickly react to these changes.
- **Generalization to Various Performance Metrics:** The prediction system should be flexible to handle various performance metrics according to the requirements of different applications, regardless of the different and diverse nature of the metrics.

These requirements create research challenges that have not been completely solved by existing systems.

1.3.3 Contributions of this Thesis

This thesis summarizes results made on developing systems that meet all the above-mentioned requirements. In particular, the following distinct contributions are made.

- **Novel Formulation based on Matrix Completion:** The problem of end-to-end network performance prediction is formulated as a **matrix completion** problem where a partially observed matrix is to be completed. Here, the matrix contains performance measures between network nodes with some of them known and the others unknown and thus to be filled. The advantages of this matrix completion formulation are threefold.

  1. It relies on neither structural information of the network nor geometric constraints. Instead, it exploits the spatial correlations across network measurements on different paths, which have long been observed in numerous research.
  2. It is generic and flexible to deal with various metrics in a unified framework, despite their diverse nature.
  3. The underlying matrix completion problem has been well studied not only in theory but also in practice, with various algorithms potentially applicable.

- **Decentralized Solution by Matrix Factorization and Stochastic Gradient Descent:** The correlations across network measurements often induce the related performance matrices to be low-rank. In observing this phenomenon, the matrix completion problem is solved by low-rank matrix factorization. In particular, a novel decentralized approach based on Stochastic Gradient Descent (SGD) is proposed, which is founded on the stochastic optimization theory with nice convergence guarantees. The so-called **Decentralized Matrix Factorization by Stochastic Gradient Descent** (**DMFSGD**) approach has two distinct features.

  1. It requires neither explicit constructions of matrices nor special nodes such as landmarks and central servers where measurements are collected and processed. Instead, by letting network nodes exchange messages with each other, matrix factorization is

collaboratively and iteratively achieved at all nodes, with each node equally retrieving a small number of measurements.
2. It is simple and computationally lightweight, containing only vector operations.

These features make DMFSGD suitable for dealing with practical problems, when deployed in real applications, such as measurement dynamics, where network measurements vary largely over time, and network churn, where nodes join and leave a network frequently.

- **Qualitative Representation of End-to-End Network Performance:** Conventionally, the performance of a network path is represented by the real value of some metric. While this quantitative representation has been commonly accepted by the networking community, it does not reflect the QoS experience perceived by end users, which by definition is what network performance is concerned with. Thus, performance measures that are more qualitative than quantitative are investigated, leading to new representation based on *binary classes* (The performance is "good" or "bad".) and on *ordinal ratings* (The performance is quantized from 1 star to 5 stars.). Such qualitative representations have several advantages.

  1. Class and rating information fulfill the requirements of applications as the goal of intelligent peer selection is to find satisfactory paths, instead of the optimal ones.
  2. Qualitative measures further reduce actual measurement costs since only coarse measurements are needed.
  3. Both classes and ratings are dimensionless or pure numbers with no unit. This feature unifies different metrics and eases their processing in applications.
  4. In addition, as both class and rating measures can be nicely integrated in the matrix completion framework, the same DMFSGD approach is applicable for their prediction, with little modification required.

  In light of these advantages, network performance prediction based on classes and ratings is also studied.

The resulting prediction system has been extensively evaluated on various publicly-available datasets of two kinds of metrics, namely RTT and ABW. These experiments support the above-mentioned features and advantages, showing not only the scalability and the accuracy of the DMFSGD approach but also its usability in real Internet applications. In addition, the benefits of predicting performance classes and ratings, rather than their actual values, are demonstrated on the task of intelligent peer selection.

1.4 Outline of this Thesis

The remainder of this thesis is organized as follows. Chapter 2 introduces related work on network performance prediction. Chapter 3 introduces some background knowledge on matrix completion and matrix factorization. Chapter 4 gives the general matrix completion framework for

network performance prediction and the basic algorithms that solve the problem by decentralized matrix factorization. Chapter 5, 6 and 7 describe the detailed algorithms and implementations for predicting network delays, binary performance classes and ordinal performance ratings respectively. Chapter 8 concludes this thesis and discusses some future work.

1.5 Publications by the Author

The work presented in this thesis has been published or submitted in the following papers.

- Yongjun Liao, Wei Du, Pierre Geurts and Guy Leduc, DMFSGD: A Decentralized Matrix Factorization Algorithm for Network Distance Prediction, accepted by IEEE/ACM Transactions on Networking on Dec. 13th 2012, DOI: 10.1109/TNET.2012.2228881.

- Wei Du, Yongjun Liao, Pierre Geurts and Guy Leduc, Ordinal Rating of Network Performance and Inference by Matrix Completion, CoRR (Arxiv), abs/1211.0447v1, 2012.

- Yongjun Liao, Wei Du, Pierre Geurts and Guy Leduc, Decentralized Prediction of End-to-End Network Performance Classes, The 7th International Conference on emerging Networking EXperiments and Technologies (CoNEXT), 6-9 Dec. 2011, Tokyo, Japan, ACM.

- Yongjun Liao, Pierre Geurts and Guy Leduc, Network Distance Prediction Based on Decentralized Matrix Factorization, IFIP Networking 2010, **best paper award**, 11-13 May 2010, Chennai, India, LNCS 6091, pp. 15-26, Springer.

In addition, the following papers have been published but not included in this thesis.

- Yongjun Liao and Guy Leduc, Triangle Inequality Violation Avoidance in Internet Coordinate Systems, Trilogy Future Internet Summer School, poster, 24-28 Aug. 2009, Louvain-la-Neuve, Belgium.

- Yongjun Liao, Mohamed Ali Kaafar, Bamba Gueye, François Cantin, Pierre Geurts and Guy Leduc, Detecting Triangle Inequality Violations in Internet Coordinate Systems by Supervised Learning (Work in Progress), IFIP Networking 2009, 12-14 May 2009, Aachen, Germany, LNCS 5550, pp. 352-363, Springer.

Chapter 2

Overview of Network Performance Prediction

Numerous approaches have been developed that predict end-to-end network performance from a few measurements, based on different assumptions and designed for different metrics. This section reviews related work which is categorized into two categories of either topology-based or model-based approaches.

2.1 Topology-Based Approaches

Topology-based approaches solve the prediction problem by utilizing the topology and routing information of the network [16, 75, 20]. The general idea is that for additive metrics such as RTT and packet loss rates (in the form of $\log(1-\text{loss rate})$), link performance can be recovered from a few known path performances by using the routing table of the network. This is possible because many links are shared across network paths due to the simple topology of the Internet core. Then, the performance of an unmeasured path can be computed by the sum of the performance of the links on that path.

In particular, suppose a network where nodes are connected by m links, forming in total n end-to-end paths. Let x be the performance vector of all links with x_j the performance of the jth link, $j = 1, \ldots, m$. Let y be the performance vector of all paths with y_i the performance of the ith path, $i = 1, \ldots, n$. Additionally, let $G \in \{0, 1\}$ be the routing matrix of $n \times m$ whose entries indicate the traversal of a given link by a given path, i.e.,

$$G_{ij} = \begin{cases} 1 & \text{if path } i \text{ traverses link } j \\ 0 & \text{otherwise} \end{cases}.$$

Thus,

$$y = Gx.$$

Normally, the number of paths n is much larger than the number of links m. This suggests that it is possible to select k paths to monitor and use those measurements to recover x. Let $\bar{y}$ be

the performance vector of the measured paths and $\bar{G}$ be the routing table of the measured paths. $\bar{G}$ is a submatrix of G, containing the rows corresponding to those measured paths. Thus,

$$\bar{y} = \bar{G}x.$$

Then,

$$x = \arg\min l(\bar{y}, \bar{G}x),$$

where l is a loss function that penalizes the difference between two variables. Different methods have been proposed to solve the above minimization problem [75].

Generally, due to the redundancy of link usage across paths, G is rank deficient [16], or more precisely, approximately low rank [20], i.e.,

$$\text{rank}(G) = r \ll \min\{m, n\}.$$

This means that the exact recovery of x is impossible [16]. Nevertheless, it has been shown that if the measured paths are carefully chosen and are sufficient, x can still be recovered with high accuracy [16, 75, 20].

Then, for an unmeasured path s,

$$y_s = g_s x,$$

where g_s is the sth row of G corresponding to the routing of path s.

Although interesting, topology-based approaches have two main shortcomings. First, they are only applicable to additive metrics such as RTT and packet loss rates, and fail for others such as ABW whose value is determined by the bottleneck link of a path. Second, acquiring and maintaining the dynamic routing matrix for a large network is very expensive and sometimes not possible.

2.2 Model-Based Approaches

Alternatively, model-based approaches solve the prediction problem by building models for network performance spaces. Generally, a model contains feature vectors associated with each node and a function by which the performance of a network path can be predicted using the feature vectors of the two end nodes. The feature vectors are model parameters that are learned by using a few measured path performance. These approaches are more practical and appealing in that they rely on no routing information of the network and some can even deal with various metrics in a unified framework.

This thesis discusses and compares two simple and general models, namely Euclidean embedding and matrix factorization.

2.2.1 Euclidean Embedding

The model of Euclidean embedding is only applicable for the delay metric of RTT. As the delays between network nodes reflect the "distance" between them, network delay is also called *network distance*. Thus, Euclidean embedding seeks to embed network nodes into a metric space

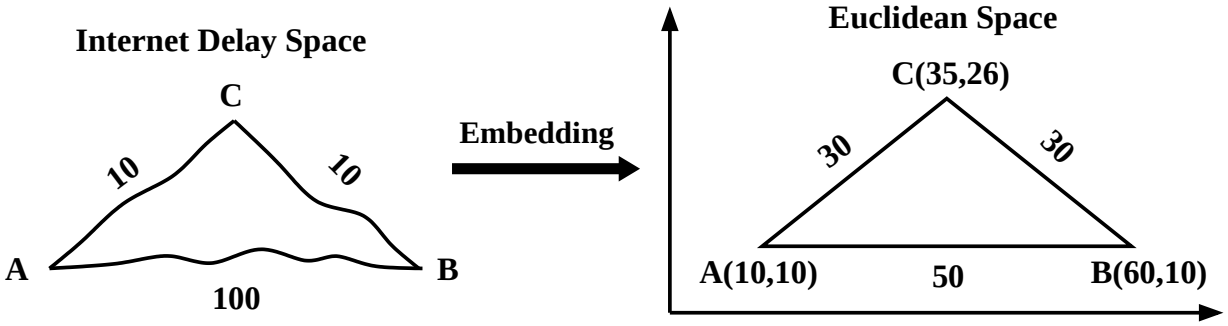


Figure 2.1: The model of Euclidean embedding. The triangle of $\triangle ABC$ violates the constraint of the Triangle Inequality as $\overline{AB} > \overline{AC} + \overline{BC}$. Thus, the embedding of node A, B and C in a coordinate system suffers from large errors.

where each node is assigned a coordinate from which distances can be directly computed. Two representatives are Global Network Positioning (GNP) [58] and Vivaldi [22].

GNP firstly proposed the idea of network embedding that relies on a small number of landmarks. Based on inter-landmark distance measurements, the landmarks are first embedded into a metric space such as Euclidean or spherical coordinate systems. Then, the ordinary nodes calculate their coordinates with respect to the landmarks. Vivaldi extended GNP in a decentralized manner by eliminating the landmarks. It simulates the network by a physical system of springs and minimizes its energy according to Hooke's law to find an optimal embedding.

In all metric spaces, distances undergo two important properties:

- Symmetry: $\mathrm{d}(A, B) = \mathrm{d}(B, A)$;
- Triangle Inequality: $\mathrm{d}(A, B) + \mathrm{d}(B, C) \geqslant \mathrm{d}(A, C)$.

However, network distances are not necessarily symmetric especially when represented by one-way delays [60, 32]. The bigger issue is the property of triangle inequality. Many studies have shown that the violations of triangle inequality (TIV) are widespread and persistent in current Internet [91, 43, 83, 22, 2, 48]. In the presence of TIVs, metric space embedding shrinks the long edges and stretches the short ones, degrading heavily the accuracy of the embedding. Figure 2.1 illustrates the idea of Euclidean embedding for network distance prediction and the impact of TIVs on the accuracy.

2.2.2 Matrix Factorization

The model of matrix factorization has also been used for network performance prediction, illustrated in Figure 2.2. The technique is well known for its success in solving the problem of matrix completion [14, 57]. One advantage of matrix factorization is that it is general and flexible to deal with not only delay metrics but also bandwidth metrics and possibly others.

The general idea is that a matrix, X, of size $n \times n$ is approximated by a low-rank matrix, $\hat{X}$, of rank r, where $r \ll n$, which is factorized into the product of two smaller matrices, U and V, of size $n \times r$, i.e.,

$$X \approx \hat{X} = UV^T.$$

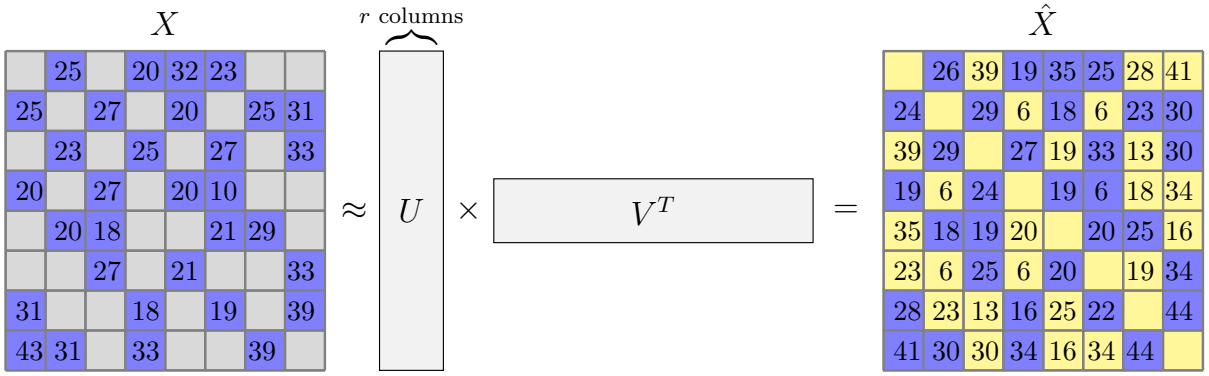


Figure 2.2: The model of matrix factorization. As matrix factorization makes no assumption of geometric constraints, it is not affected by the widespread and persistent TIVs in the Internet delay space.

After the factorization, each node i is associated with two vectors, u_i and v_i, corresponding to the ith row of U and of V. Then, the prediction of the performance of the path from node i to node j is computed by the dot product of u_i and v_j. The details of how matrix factorization is applied for network performance prediction will be given in the following chapters.

Matrix factorization was firstly used for network distance prediction to overcome the shortcomings of Euclidean embedding. The first system was Internet Distance Estimation Service (IDES) [52] which has the same landmark-based architecture as GNP. IDES factorizes a small but full inter-landmark distance matrix, at a so-called information server, by using Singular Value Decomposition (SVD). Similarly, Phoenix treated the early-entered nodes as landmarks and allowed an ordinary node to select any existing nodes in the system which already have coordinates assigned [18]. Landmark-based systems suffer from several drawbacks including single-point failures, landmark overloads, and potential security problems. The selection of landmarks can also affect the accuracy of the prediction. Like Vivaldi, our work in [46] eliminated the landmarks and solved the matrix factorization problem by Alternating Least Square (ALS) in a fully decentralized manner.

Later, it was realized that matrix factorization captures the correlations between matrix entries which exist for other metrics such as bandwidth. Our work in [44] then extended matrix factorization for predicting bandwidth. Comparing with other approaches, the ability of dealing with various metrics in a unified framework and of making prediction without additional information on the network makes matrix factorization appealing and unique. Thus, the rest of the thesis focuses on the model of matrix factorization for network performance prediction, including the algorithmic solutions and the usability by applications.

2.3 Other approaches

Besides the above-mentioned approaches, other approaches and models have been proposed for network performance prediction and can be found in [64, 59, 51], among many others. Below, a brief overview on these approaches is given.

Sequoia [64] exploits the observation that "the Internet is tree-like" and builds explicitly a tree graph for the Internet. That is, network nodes are embedded to a "virtual prediction tree", and the edges of the tree are annotated with some metric of network performance. The performance of the path between two nodes is estimated as the length of the path on the tree connecting them. Unlike Euclidean embedding, Sequoia is applicable not only to RTT but also to available bandwidth.

BRoute [59] is another system for available bandwidth prediction. The design of BRoute is based on the observation that more than 86% bottleneck links of network paths are within 4 hops from end nodes, called *path edges*. This suggests that the bandwidth information for path edges can be used to infer end-to-end available bandwidth with high probability. Based on this observation, BRoute identifies the bottleneck links of network paths by using network management tools such as traceroute and the BGP routing information.

iPlane [51] is a system that predicts Internet path performance such as RTT, available bandwidth and loss rate. It builds an annotated map of the Internet by using both the active measurement tools and the router-level topology, and predicts end-to-end performance by composing measured performance of segments of known Internet paths.

These approaches are interesting in different aspects, but they either build explicitly a structural model for the Internet or rely on the structural information of the network which is assumed unavailable in the approaches developed in this thesis.

Chapter 3

Fundamentals of Matrix Completion and Matrix Factorization

Two major contributions of the thesis are the formulation of network performance prediction as a matrix completion problem and its decentralized resolution by matrix factorization. Thus, this chapter briefly introduces some basic knowledge on matrix completion and matrix factorization.

3.1 Matrix Completion

Matrix completion addresses the problem of recovering a low-rank matrix from a subset of its entries. Intuitively, such completion is possible, because a low-rank matrix contains redundancies and thus not all of its entries are needed to represent it. For example, consider a $n \times n$ matrix, denoted by X, of rank r, where $r \ll n$. While X has n^2 entries, it only has $(2n - r)r$ degrees of freedoms [14]. In particular, the theory of matrix completion shows that under some suitable conditions, with high probability, X can be exactly recovered from just $O(n^b r \log n)$ randomly observed entries [14], where b is only slightly larger than 1. In practice, X is often full rank but with a rank r dominant component. That is, X has only r significant singular values and the others are negligible. In such cases, a matrix of rank r, denoted by $\hat{X}$, can still be found, using about the same amount of entries, that approximates X with high accuracy [13, 38].

3.1.1 Recommender Systems and Netflix Prize

The research on matrix completion was largely motivated by the demands of recommender systems which seek to predict a preference measure that a user would give to an item (such as music, books, or movies) or a social element (e.g. people or groups) [39]. Such function enables e-commerce companies such as Amazon and Netflix to provide personalized services and match consumers with more appropriate products.

The problem in recommender systems is clearly a matrix completion problem, where the matrix to be completed is a preference matrix with the ijth entry representing the preference of the ith user to the jth item. Each user provides the preferences to a few items, from which the miss-

ing preferences of users to items are predicted. Table 3.1 shows an example of a recommender system as matrix completion. In the field of machine learning, the technique for recommender systems is also called collaborative filtering, as making recommendations amounts to predicting (filtering) the interests of a user by collecting preference information from other users (collaborating with other users) [79, 70].

Table 3.1: An example of a recommender system as matrix completion.

	item1	item2	item3	item4	item5	item6	item7
user1	5	3	4	1	?	?	?
user2	5	3	4	1	5	2	5
user3	5	?	4	1	5	3	?
user4	1	3	2	5	1	4	2
user5	4	?	4	4	4	?	4

One of the most famous recommender systems or collaborative filtering algorithms is the one at Netflix [56], an American online DVD-rental company. The Netflix system deals with the preferences of ordinal ratings on a scale of $\{1, 5\}$. In 2006, Netflix released an open competition for algorithms that predict user ratings for films, based on previous ratings without any other information about the users and films [57, 4]. A grand prize of US$\$1,000,000$ was to be given to the first algorithm which could improve the accuracy of Netflix's own algorithm, Cinematch, by 10%. The Netflix prize drew intensive attentions of the machine learning community and greatly promoted the development of not only the theory but also the practice of matrix completion, leading to various engineering and algorithmic solutions. In 2009, the prize was given to the team of BellKor's Pragmatic Chaos as it achieved a 10.06% improvement.

3.1.2 Low-Rank Approximation

Mathematically, matrix completion is formulated as the low-rank approximation that solves the following minimization problem

$$\begin{array}{ll} \text{minimize} & \text{Rank}(\hat{X}). \\ \text{subject to} & L(P_\Omega(X), P_\Omega(\hat{X})) \leqslant \delta \end{array} \tag{3.1}$$

Ω is the set of observed entries and P_Ω is a sampling function that preserves the entries in Ω and turns the others into 0, i.e.,

$$[P_\Omega(X)]_{ij} = \begin{cases} x_{ij} & (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

where x_{ij} is the ijth entry of X. The function L penalizes the entry-wise difference between two matrices, i.e.,

$$L(X, \hat{X}) = \sum_{i,j=1}^{n} l(x_{ij}, \hat{x}_{ij}), \tag{3.2}$$

with l a loss function that takes on various forms. For example, the l_2 or square loss function is the most widely used and defined as

$$l(x, \hat{x}) = (x - \hat{x})^2. \tag{3.3}$$

In words, matrix completion searches for a low-rank matrix $\hat{X}$ that approximates X with sufficient accuracy at the observed entries in Ω. The missing entries in X, i.e. those (i, j)s that are not in Ω, are predicted by the corresponding entries in $\hat{X}$.

3.1.3 Low-Norm Approximation

Generally, the rank function is difficult to process, since it is neither convex nor continuous. This difficulty can be addressed by the convex relaxation that replaces the rank by the trace norm [14], leading to the low-norm approximation that solves the following minimization problem

$$\begin{aligned} &\text{minimize} \quad \text{Trace}(\hat{X}). \\ &\text{subject to} \quad L(P_\Omega(X), P_\Omega(\hat{X})) \leqslant \delta \end{aligned} \tag{3.4}$$

The trace norm, Trace(X), is the sum of the singular values of X, and is the tightest convex surrogate of the rank. The advantage of such relaxation is that the trace norm is convex and its minimization can be solved by a general semidefinite programming (SDP) solver, with the guarantee of finding the globally optimal solution.

Interesting as it is, recent studies show that the low-norm approximation relies on rather strong assumptions. For example, it assumes that X is indeed low-rank with $r = O(\log n)$, that X has to obey certain incoherence properties, and that the observed entries in Ω are sampled uniformly at random. In addition, as the low-norm approximation searches for $\hat{X}$ in the $n \times n$ matrix space, it tends to be computationally demanding and does not scale well.

3.2 Matrix Factorization

Although difficult, the low-rank approximation can still be tackled directly by adopting some compact representation of low-rank matrices. For example, eq. 3.1 can be reformulated as the following dual problem

$$\begin{aligned} &\text{minimize} \quad L(P_\Omega(X), P_\Omega(\hat{X})). \\ &\text{subject to} \quad \text{Rank}(\hat{X}) \leqslant r \end{aligned} \tag{3.5}$$

As $\text{rank}(\hat{X}) \leqslant r$, $\hat{X}$ can be factorized, i.e.,

$$\hat{X} = UV^T, \tag{3.6}$$

where U and V are matrices of size $n \times r$. Thus, we can look for the pair (U, V), instead of $\hat{X}$, by solving

$$\text{minimize} \quad L(P_\Omega(X), P_\Omega(UV^T)). \tag{3.7}$$

Note that the above minimization has no unique solution as

$$\hat{X} = UV^T = (UG)(VG^{-T})^T, \tag{3.8}$$

where G is any arbitrary $r \times r$ invertible matrix and $G^{-T} = (G^{-1})^T$. Thus, the pair (U, V) is equivalent to the pair (UG, VG^{-T}) in that they produce the same $\hat{X}$.

The non-uniqueness of the factorization in eq. 3.8 may create numerical problems such as overflows. That is, an ill-posed solution can potentially be found, with one of the U and V having a very big norm and the other very small. A common way to overcome this problem is through regularization that penalizes the norms of the solutions, resulting in the following regularized objective function,

$$\text{minimize} \quad L(P_\Omega(X), P_\Omega(UV^T)) + \lambda||U||_2^2 + \lambda||V||_2^2. \tag{3.9}$$

where λ is the regularization coefficient that controls the extent of regularization. $||\bullet||_2^2$ is the l_2 or Frobenius norm of a matrix, defined as

$$||X||_2^2 = \sum_i \sum_j x_{ij}^2. \tag{3.10}$$

Thus, the regularization improves the numerical stability by forcing to choose, among the infinite number of pairs of (U, V) which produce the same $\hat{X}$, the one with the smallest norm. Besides, it also helps overcome a well-known problem called overfitting in the field of machine learning. In words, a "perfect" model could be learned, with no or small errors on the training data in Ω but large errors on the missing data which are not used during learning.

To make the expression simpler and for the reason that will become clear later, eq. 3.9 is rewritten as the following

$$\text{minimize} \quad \sum_{(i,j)\in\Omega} l(x_{ij}, u_i v_j^T) + \lambda \sum_{i=1}^{n} u_i u_i^T + \lambda \sum_{i=1}^{n} v_i v_i^T, \tag{3.11}$$

where u_i and v_i are the ith row of U and V respectively, and

$$u_i v_j^T = \hat{x}_{ij} \approx x_{ij}.$$

While eq. 3.9 and eq. 3.11 are equivalent, the latter contains no matrices. In doing so, we can look for (u_i, v_i)s, $i = 1, \ldots, n$, instead of (U, V), by using a set of x_{ij}s, $(i, j) \in \Omega$, instead of X. This difference enables a more efficient implementation of matrix factorization which can be decentralized for network applications.

This class of techniques is generally called low-rank matrix factorization. Unlike the low-norm approximation in eq. 3.4, the low-rank approximation in eq. 3.7, 3.9 or 3.11 is non-convex and thus its minimization only produces locally optimal results. However, as the pair (U, V) has $2nr$ entries in contrast to the n^2 for $\hat{X}$, matrix factorization is much more appealing for large-scale matrix completion problems. In addition, it relies less on those conditions required by the low-norm approximation and often works better when dealing with real noisy data. Thus, matrix factorization has been successfully used in solving various engineering problems. This thesis shows that it is also applicable to the problem of network performance prediction.

3.3 Algorithms of Matrix Completion

Numerous algorithms of matrix completion have been proposed, based on either low-norm approximation or low-rank factorization. Those of low-norm approximation include semidefinite programming [25], singular value thresholding [12], MultiRegression [36] and SOFT-IMPUTE [53], etc. Those of low-rank factorization include LMaFit [86], GECO [73], OptSpace [38] and ADMiRA [42], etc. In addition, [10] compared different optimization schemes such as gradient descent and Newton algorithms for solving the low-rank factorization problem. While interesting in different aspects, these approaches and studies focused on the centralized resolution where the data is collected and processed at a central server.

For very large-scale matrix completion problems, two simple matrix factorization algorithms are particularly interesting, namely **alternating least squares (ALS)** and **stochastic gradient descent (SGD)** [79, 39]. For example, they have been extensively used in solving the problem of the Netflix prize which requires to complete a matrix of ratings from $480,189$ users to $17,770$ movies. Moreover, ALS and SGD can be implemented in a parallel and decentralized manner, which makes them appealing for networking applications.

Below, a detailed introduction of ALS and SGD for matrix completion is given. Recall that the goal is to find (u_i, v_i)s, $i = 1, \ldots, n$ by minimizing eq. 3.11. To be concrete, it is assumed that the l_2 loss function is used in eq. 3.11.

3.3.1 Alternating Least Squares

Eq. 3.11 is non-convex with respect to the unknowns (u_i, v_i)s, $i = 1, \ldots, n$. However, if we fix all but one unknowns, the optimization problem becomes a standard least-squares problem and can be solved analytically and optimally. For example, suppose only u_i is unknown, then

$$u_i = \arg\min \sum_{\text{all } j\text{s if } (i,j)\in\Omega} (x_{ij} - u_i v_j^T)^2 + \lambda u_i u_i^T. \tag{3.12}$$

Similarly, if only v_i is unknown, then

$$v_i = \arg\min \sum_{\text{all } j\text{s if } (j,i)\in\Omega} (x_{ji} - u_j v_i^T)^2 + \lambda v_i v_i^T. \tag{3.13}$$

Thus, alternating least squares (ALS) rotate between fixing the u_is and fixing the v_is. At each step, only one of the u_is and v_is is treated as unknown and updated. This ensures that each step decreases the loss in eq. 3.11 until convergence.

3.3.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a variation of traditional batch gradient descent and is often used for online machine learning [8]. In SGD, instead of collecting all training samples beforehand and computing the gradients over them, each iteration of SGD chooses one training sample

at random and updates the parameters being estimated along the negative gradients computed over that chosen sample.

In using SGD for solving eq. 3.11, the algorithm loops through all x_{ij}s, $(i,j) \in \Omega$. For each x_{ij}, the system updates u_i and v_j so that $x_{ij} \approx u_i v_j^T$, which is done by reducing the loss

$$l_{ij} = (x_{ij} - u_i v_j^T)^2 + \lambda u_i u_i^T + \lambda v_j v_j^T. \tag{3.14}$$

The gradients are given by

$$\frac{\partial l_{ij}}{\partial u_i} = -(x_{ij} - u_i v_j^T) v_j + \lambda u_i, \tag{3.15}$$

$$\frac{\partial l_{ij}}{\partial v_j} = -(x_{ij} - u_i v_j^T) u_i + \lambda v_j. \tag{3.16}$$

Note that a factor 2 is dropped from the derivatives of the L_2 loss function and from the regularization terms for mathematical convenience. Then, u_i and v_j are updated by a magnitude proportional to a learning rate η in the opposite direction of the gradients, yielding

$$u_i = (1 - \eta\lambda) u_i + \eta (x_{ij} - u_i v_j^T) v_j, \tag{3.17}$$

$$v_j = (1 - \eta\lambda) v_j + \eta (x_{ij} - u_i v_j^T) u_i. \tag{3.18}$$

These updates are iterated until convergence.

SGD has been popularized due to its ease of implementation and a fast running time. It is particularly suitable for network performance prediction as network measurements can be acquired on demand and processed locally at each node. It also has simple update rules that involve only vector operations and is able to deal with large-scale dynamic measurements. These features will be justified in the following chapters.

Chapter 4

Network Performance Prediction as Matrix Completion

This chapter introduces the formulation of network performance prediction as matrix completion and how to solve it by matrix factorization in a fully decentralized manner.

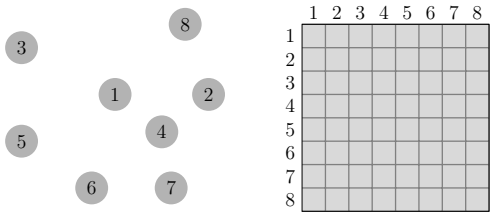
4.1 A Matrix Completion View

The problem of network performance prediction can be viewed as a matrix completion problem. Here, the matrix to be completed, X, is a performance matrix, with the ijth entry, x_{ij}, representing the performance of the path from node i to node j, measured by a chosen metric such as RTT and ABW (available bandwidth). Each node probes a few other nodes, measures the performance of the paths between them. The measurements are put at the corresponding entries of X, and the missing entries are the performances of those unmeasured paths and need to be predicted. The process is illustrated in Figure 4.1 with an example of a network of 8 nodes.

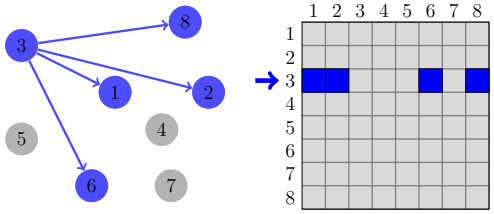
4.1.1 Feasibility and Low-Rank Characteristic

In order for network performance prediction to be feasible, the performance measurements on different paths have to be correlated. Otherwise, the inference of unmeasured performance from a few measurements would be impossible or suffer from large errors.

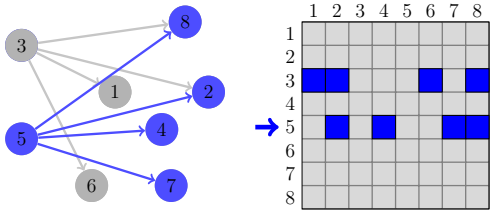
The correlations among network measurements exist, because Internet paths with nearby end nodes often overlap or share bottleneck links due to the simple topology of the Internet core. The redundancy of link usage across paths causes the performance of many paths to be dependent of each other. For example, the congestion at a certain link causes the delays of all paths that traverse this link to increase jointly. These correlations induce the related performance matrix to be low-rank and enable the inference problem to be solved by matrix completion via matrix factorization. The low-rank characteristic of performance matrices of RTT and of ABW is illustrated by the spectral plots in Figure 4.2. It can be seen that the singular values of these matrices decrease fast, which is an empirical justification of the low-rank characteristic.



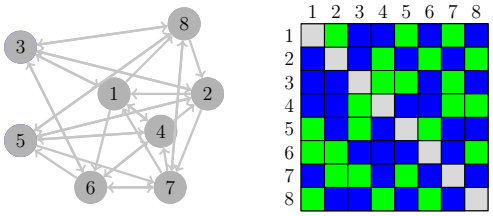
(a) A network of 8 nodes.



(b) Node 3 probes node 1, 2, 6 and 8.



(c) Node 5 probes node 2, 4, 7 and 8.



(d) A matrix with missing entries is formed.

Figure 4.1: A matrix completion view of network performance prediction. In the constructed matrix in (d), blue entries contain measured performance and green entries are missing and are to be filled. Note that the diagonal entries are empty as they are the performance of a node to itself and of no interest.

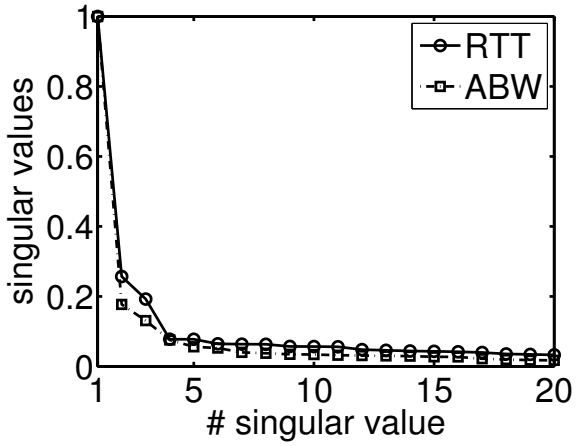


Figure 4.2: The singular values of a 2255×2255 RTT matrix, extracted from the Meridian dataset [87], and a 201×201 ABW matrix, extracted from the HP-S3 dataset [89]. The singular values are normalized so that the largest ones are equal to 1.

The above insights suggest that the low-rank phenomenon is likely to be general and exists in a wide variety of scenarios, which has also been confirmed in various researches [20, 80, 16, 45, 44]. While the low-rank characteristic enables a network performance matrix X to be approximated by a low-rank matrix $\hat{X}$ of rank r, the value of r can not be determined a priori and depends on a number of network properties including network infrastructure and usage, routing algorithms and node distributions, etc. Thus, the rank r is treated as a parameter and studied empirically for a given dataset.

4.1.2 Connections to Recommender Systems

The problem of network performance prediction bears strong similarities to the problem of recommender systems. Consider that network nodes are users and each node treats other nodes as items. The performance of a network path reflects a preference measure of how one end node would like to contact the other end node. Then, the task of intelligent path and peer selection which is required by many Internet applications such as CDNs or P2P downloading amounts to recommending, for a node, the paths and peers that are satisfactory, for example, with a small delay or high ABW. This insight suggests that many collaborative filtering algorithms developed for recommender systems can potentially be applied for the problem of network performance prediction.

4.2 Decentralized Prediction of Network Performance

It is natural to require that the prediction system is integrated in network applications which usually have a decentralized architecture. A fully decentralized prediction is enabled by the

low-rank matrix factorization formulated as eq. 3.11. Recall that the objective is

$$\{(u_i, v_i), i = 1, \ldots, n\} = \arg\min \sum_{(i,j)\in\Omega} l(x_{ij}, u_i v_j^T) + \lambda \sum_{i=1}^{n} u_i u_i^T + \lambda \sum_{i=1}^{n} v_i v_i^T. \tag{4.1}$$

u_is and v_is are row vectors of length r, where r is the effective rank of X.

The requirement of decentralized prediction puts constraints on the system design:

- No matrix is explicitly constructed.
- Path measurements are probed and processed locally at each node.

To meet these constraints, the following measures are taken in the system design:

- (u_i, v_i)s, $i = 1, \ldots, n$ are distributively stored, i.e., (u_i, v_i) is stored at node i. (u_i, v_i) is called the *coordinate* of node i.
- Each node selectively probes a number of other nodes, called *neighbors*. Denote $\mathcal{N}_i$ the neighbor set of node i, $i = 1, \ldots, n$.
- Each node updates its coordinate by minimizing local losses related to the probed measurements between the node and its neighbors.
- The unmeasured performance is estimated from node coordinates by

$$\hat{x}_{ij} = u_i v_j^T. \tag{4.2}$$

- The system architecture integrates the prediction algorithm and the measurement methodology for the chosen metric.

In particular, the neighbors of each node can generally be randomly selected from the set of available nodes in the network. The system architecture needs to be modified for different metrics due to their different measurement methodologies. For example, the measurement of RTT is probed and computed by the sender, whereas that of ABW is probed by the sender but computed by the receiver. This difference needs to be addressed in the design of the decentralized system architecture.

4.3 Decentralized Matrix Factorization by Stochastic Gradient Descent

As mentioned in Section 3.3.2 in Chapter 3, stochastic gradient descent (SGD) is particularly suitable for solving the matrix factorization problem and is thus adopted for decentralized network performance prediction. The approach is called *Decentralized Matrix Factorization by Stochastic Gradient Descent (DMFSGD)*. The general idea is that when a measurement x_{ij} becomes available, node i and node j collaborate to update their coordinates so that the loss

$$l(x_{ij}, u_i v_j^T) + \lambda u_i u_i^T + \lambda v_j v_j^T \tag{4.3}$$

is reduced and that $u_i v_j^T = \hat{x}_{ij}$ approximates x_{ij} better. The updates are along the opposite direction of the gradient of the above loss. Figure 4.3 illustrates the process of stochastic gradient descent for matrix factorization. DMFSGD takes into account how the measurement is probed. Below, how DMFSGD is applied for RTT and for ABW is introduced.

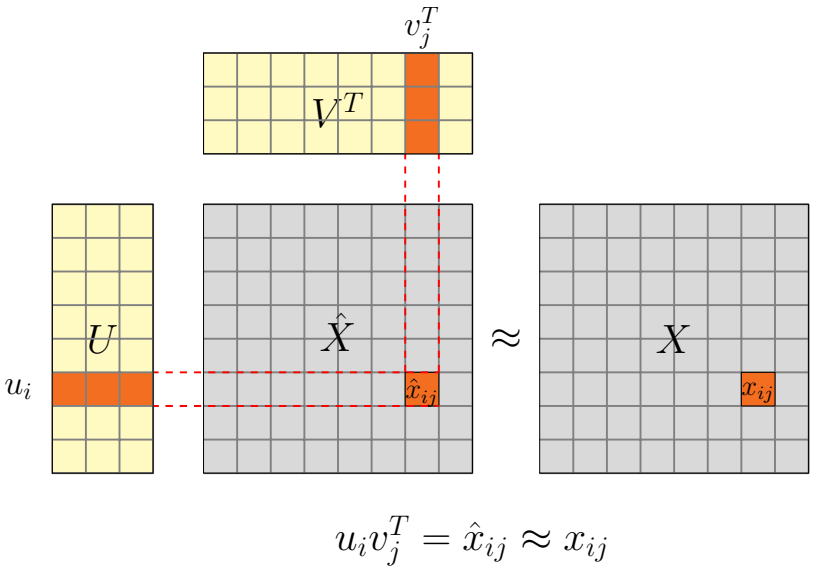


Figure 4.3: Stochastic gradient descent for matrix factorization. When a measurement x_{ij} becomes available, u_i, the ith row of U, and v_j, the jth row of V can be updated so that $u_i v_j^T = \hat{x}_{ij} \approx x_{ij}$.

4.3.1 DMFSGD for RTT

For RTT, the measurement x_{ij} is probed by and available at node i, which means that x_{ij} cannot be used by node j. However, as RTT is considered symmetric, $x_{ij} = x_{ji}$. Then, both u_i and v_i can be updated by using x_{ij} so that

$$\begin{aligned} u_i v_j^T = \hat{x}_{ij} &\approx x_{ij} \\ u_j v_i^T = \hat{x}_{ji} &\approx x_{ij}. \end{aligned}$$

Note that although the predicted RTT is not symmetric in general, i.e. $\hat{x}_{ij} \neq \hat{x}_{ji}$, the difference is small as both $\hat{x}_{ij}$ and $\hat{x}_{ji}$ approximate x_{ij}.

Hence, define the losses related to u_i and v_i at node i as

$$\begin{aligned} &l(x_{ij}, u_i v_j^T) + \lambda u_i u_i^T, \\ &l(x_{ij}, u_j v_i^T) + \lambda v_i v_i^T. \end{aligned}$$

The update rules of u_i and v_i, at node i, based on gradient descent are

$$u_i = (1 - \eta\lambda)u_i - \eta \frac{\partial l(x_{ij}, u_i v_j^T)}{\partial u_i}, \tag{4.4}$$

$$v_i = (1 - \eta\lambda)v_i - \eta \frac{\partial l(x_{ij}, u_j v_i^T)}{\partial v_i}. \tag{4.5}$$

where η, called *learning rate* or *step size*, controls the speed of the updates. η needs to be set appropriately, because a too large η results in large steps of updates and may overflow the solution, whereas a too small η makes the convergence of the algorithm slow.

4.3.2 DMFSGD for ABW

For ABW, the measurement x_{ij} is probed by node i but inferred at node j. As ABW is largely asymmetric, the symmetry trick used above for RTT is not applicable for ABW. Thus, when x_{ij} becomes available at node j, node j needs to send x_{ij} to node i so that u_i of node i and v_j of node j can both be updated.

Hence, define the losses related to u_i at node i and related to v_j at node j as

$$l(x_{ij}, u_i v_j^T) + \lambda u_i u_i^T,$$
$$l(x_{ij}, u_i v_j^T) + \lambda v_j v_j^T.$$

The update rules of u_i and v_j, at node i and node j respectively, based on gradient descent are

$$u_i = (1 - \eta\lambda)u_i - \eta \frac{\partial l(x_{ij}, u_i v_j^T)}{\partial u_i}, \tag{4.6}$$

$$v_j = (1 - \eta\lambda)v_j - \eta \frac{\partial l(x_{ij}, u_i v_j^T)}{\partial v_j}. \tag{4.7}$$

4.3.3 L_2 Loss Function

Although the loss function in eqs. 4.4 and 4.5 and in eqs. 4.6 and 4.7 can take various forms, the L_2 or square loss function is the most widely-used and given in eq. 3.3.

The gradients of the L_2 loss function in eqs. 4.4 and 4.5 and in eqs. 4.6 and 4.7 are given below. Without causing any confusion, the subscripts are dropped.

$$\frac{\partial l(x, uv^T)}{\partial u} = -(x - uv^T)v, \tag{4.8}$$

$$\frac{\partial l(x, uv^T)}{\partial v} = -(x - uv^T)u. \tag{4.9}$$

Note that the factor 2 is dropped from the derivatives of the L_2 loss function for mathematical convenience.

Other loss functions such as the robust L_1 loss function and the classification loss functions will be discussed in the following chapters.

4.3.4 Basic Algorithms

The basic DMFSGD algorithms for RTT and for ABW are given in Algorithm 1 and in Algorithm 2. Essentially, DMFSGD integrates, at each node, an inference module that performs SGD updates for matrix factorization and a measurement module that collects measurements. In the measurement module, message exchanges are incorporated and adapted for different metrics due to their different measurement methodologies. For RTT, the senders perform the coordinate updates and the receivers only response to the probes and attach their coordinates in the reply. For ABW, both the senders and the receivers perform coordinate updates and exchange coordinates.

Algorithm 1 DMFSGD_RTT(i, j)

1: node i probes node j for the RTT;
2: node j replies and sends u_j and v_j to node i;
3: node i infers x_{ij} when receiving the reply;
4: node i updates u_i and v_i by eqs. 4.4 and 4.5;

Algorithm 2 DMFSGD_ABW(i, j)

1: node i probes node j for the ABW and sends its u_i;
2: node j infers x_{ij} when probed;
3: node j replies and sends x_{ij} and v_j to node i;
4: node j updates v_j by eq. 4.7;
5: node i updates u_i by eq. 4.6 when receiving the reply;

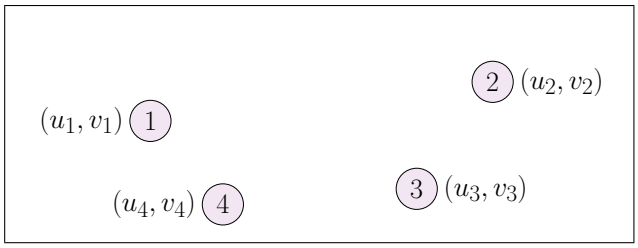
In addition, the receivers also need to send the ABW measurements to the senders. Figure 4.4 and 4.5 illustrate how DMFSGD works for RTT and for ABW with an example of a network of 4 nodes. Figure 4.6 illustrates how the performance of a network path is computed from the coordinates of the two end nodes. Note that the algorithms are generic and can be easily adapted to other metrics by taking into account their measurement methodologies. For example, one-way delays and packet loss rates are asymmetric and their prediction can be done directly by using DMFSGD for ABW.

In the prediction system, each node randomly and independently chooses a neighbor set of k nodes as references and randomly probes one of its neighbors at each time. The coordinates of the nodes are initialized with random numbers uniformly distributed between 0 and 1. Empirically, the DMFSGD algorithms are insensitive to the random initialization of the coordinates as well as the random selection of the neighbors.

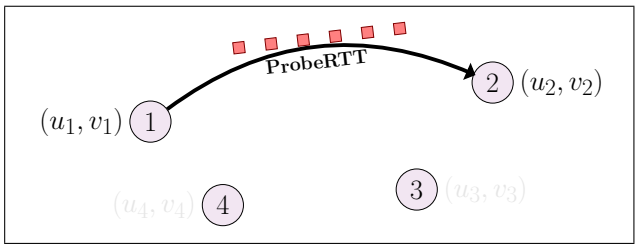
The DMFSGD algorithms and the prediction system have the following parameters:

- **neighbor number** k;
- **rank** r;
- **learning rate** η;
- **regularization** λ.

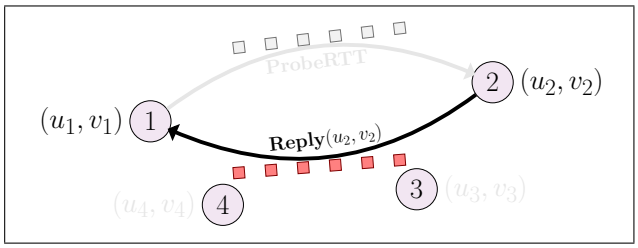
The following chapters will describe in detail the applications of the DMFSGD algorithms for RTT and for ABW and the impacts of the above parameters.



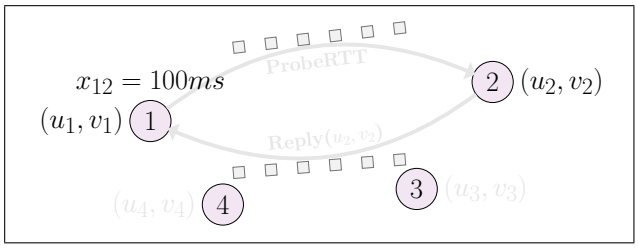
(a) A network of 4 nodes.



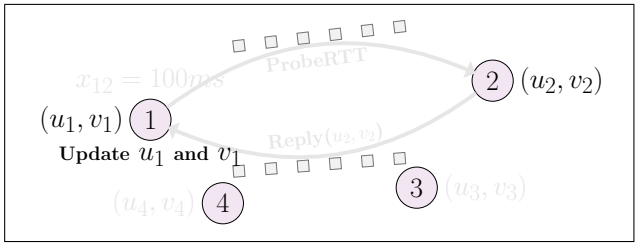
(b) Node 1 probes node 2 for the RTT.



(c) Node 2 replies and sends u_2 and v_2 to node 1.

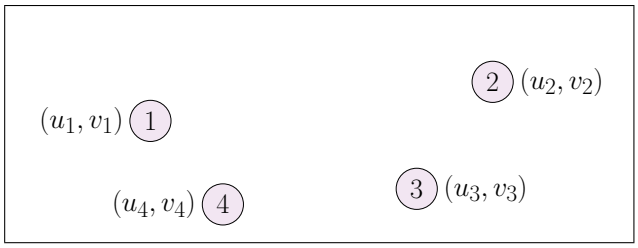


(d) Node 1 infers x_{12} when receiving the reply.

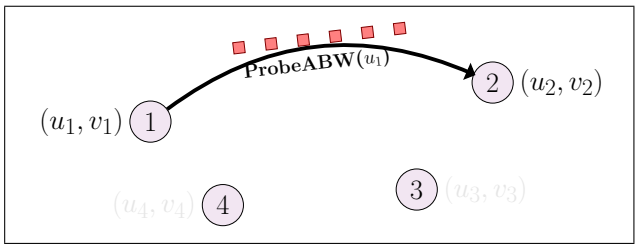


(e) Node 1 updates u_1 and v_1.

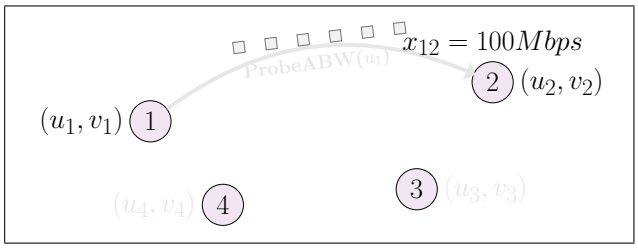
Figure 4.4: An example that shows how DMFSGD works for RTT.



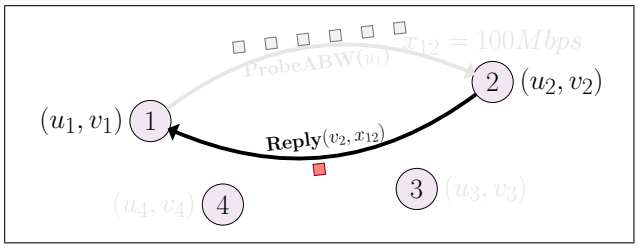
(a) A network of 4 nodes.



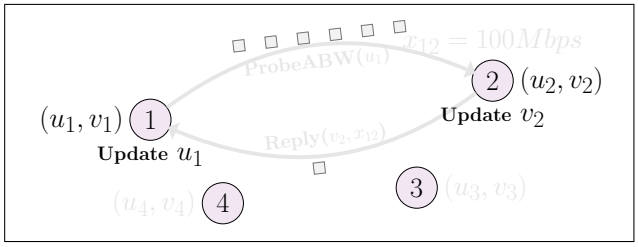
(b) Node 1 probes node 2 for the ABW and sends its u_1.



(c) Node 2 infers x_{12} when probed.



(d) Node 2 replies and sends x_{12} and v_2 to node 1.



(e) Node 1 updates u_1 and node 2 updates v_2.

Figure 4.5: An example that shows how DMFSGD works for ABW.

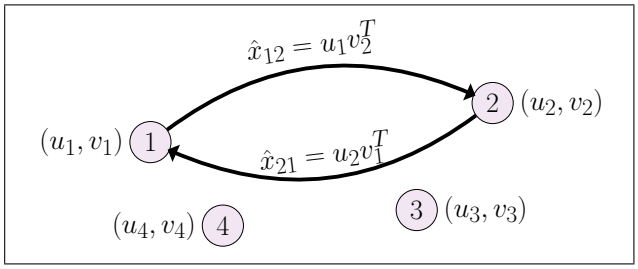


Figure 4.6: An example that shows how a node infers the performance, either RTT or ABW, of the paths connected to another node. Here, node 1 infers $\hat{x}_{12}$ and $\hat{x}_{21}$ by using its coordinate (u_1, v_1) and by retrieving the coordinate of node 2 (u_2, v_2).

Chapter 5

Predicting End-to-End Network Distance

In the networking field, delay metrics are also called "network distances" as they reflect the distances, in the unit of time (second or millisecond), between network nodes. This chapter introduces the extensions of the basic DMFSGD algorithm to network distance prediction. These extensions address the practical issues when deploying DMFSGD in real applications, including

- the minibatch and line search strategies that address the difficult choice of the learning rate parameter and the problem of network churns where nodes joint and leave a network frequently;
- robust matrix factorization that addresses the measurement noise and outliers by incorporating the robust L_1 loss function;
- non-negative matrix factorization that preserves the non-negativity of the distance.

In addition, a comprehensive comparison of matrix factorization and Euclidean embedding is provided that reveals the suitability of matrix factorization for network distance prediction. A unified view of the two models is given and leads to a unified optimization framework to solve both of them.

5.1 Network Distance

The knowledge of end-to-end network distance is important for many network applications such as P2P file sharing and CDNs. While the distance between two network nodes can be represented by either round-trip time (RTT) or one-way delay, RTT is more widely used due to its ease of acquisition by the ping-type tools. In this chapter, only the RTT metric is considered and referred to as network distance.

Network distance or RTT has the following properties.

- RTT is symmetric, as mentioned earlier in Section 1.2.1 in Chapter 1.
- RTT can sometimes be obtained passively during data transfer, without generating extra traffic. This passivity is enforced in applications such as Azureus (a P2P file sharing software) [82, 40].

- RTT is very noisy and highly dynamic. Figure 5.1 shows the RTT measurements collected from a network path in 3 days.
- RTT distributions of different networks vary largely, shown in Figure 5.2.

These properties create some challenges in network distance prediction.

- Large measurement noises can easily ruin the prediction results.
- The passive measurements on different paths can be available at uneven frequencies, i.e., some paths are measured more often than others.
- The dynamics and the large variations of the measurements make the prediction system sensitive to the parameter setting.

These challenges are addressed by some special treatments described in Section 5.3.

5.2 Vivaldi

Vivaldi [22] is an Euclidean embedding approach to network distance prediction and is arguably the state of the art in terms of accuracy and practicability. As Vivaldi is used as the baseline system and compared with DMFSGD, a brief introduction is given below.

5.2.1 Algorithm

Vivaldi assigns each network node synthetic coordinates in a coordinate space so that the distances in the coordinate space accurately predict the RTTs. This is done by simulating a network as a physical spring system, where network nodes are connected by springs.

Let u_i be the Euclidean coordinate of node i, $i = 1, \ldots, n$. The predicted distance between node i and node j is calculated by the Euclidean distance function, defined as

$$\hat{x}_{ij} = \sqrt{(u_i - u_j)^T (u_i - u_j)}. \tag{5.1}$$

Then, the energy of the spring system is given by

$$E = \sum_{(i,j) \in \Omega} (x_{ij} - \hat{x}_{ij})^2. \tag{5.2}$$

Recall that Ω is the set of measured paths where network distances are known. Thus, Vivaldi seeks for the optimal embedding by releasing this energy as much as possible.

In Vivaldi, the energy minimization is solved in a fully decentralized manner by letting each spring release its energy according to the Hooke's law. Let F_{ij} be the force that the spring between node i and node j exerts on node i, defined as

$$F_{ij} = (x_{ij} - \hat{x}_{ij}) \times u(u_i - u_j), \tag{5.3}$$

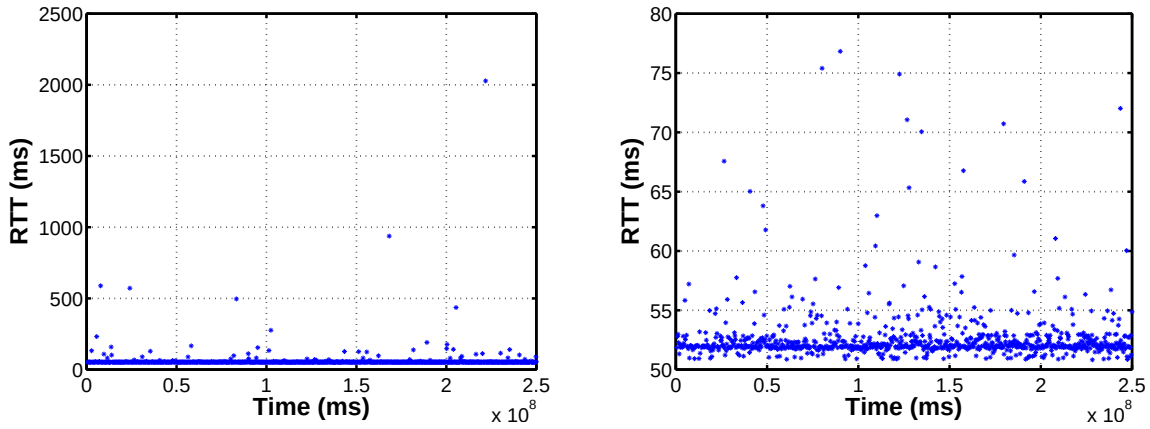


Figure 5.1: RTTs between a pair of nodes measured for 972 times in 72 hours [40]. The measurements were collected passively from Azureus. The right plot is the closeup of the left plot. Although the mean RTT of the 972 measurements is $60.36ms$, individual RTTs can go, although rarely, as large as more than $2000ms$.

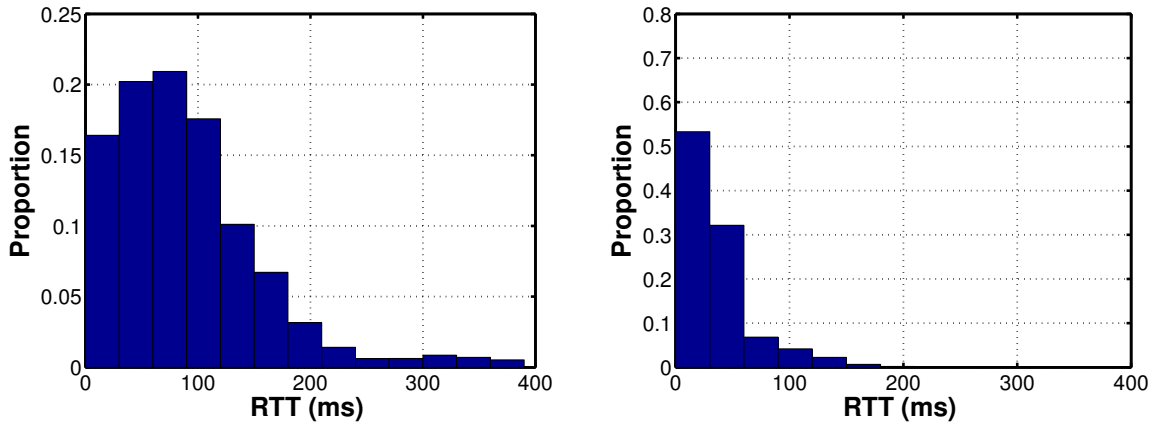


Figure 5.2: RTT distributions of the P2PSim dataset [30], on the left, and the Meridian dataset [87], on the right. It can be seen that the values of RTTs are distributed more evenly for P2PSim than for Meridian in which there are even 0.06% edges longer than $400ms$ with the largest one about $1500ms$.

where $(x_{ij} - \hat{x}_{ij})$ is the magnitude of the force and $u(u_i - u_j) = \frac{u_i - u_j}{\hat{x}_{ij}}$ is a unit vector and gives the direction of the force. Thus, node i updates its coordinate u_i to reduce the energy on the spring between node i and node j by

$$u_i = u_i + \eta \times F_{ij}. \tag{5.4}$$

where η is called *timestep* which corresponds to the learning rate in DMFSGD. All nodes collaboratively update their coordinates as above until the energy of the spring system is minimized.

5.2.2 Adaptive Timestep

Like DMFSGD, Vivaldi is sensitive to the timestep η. A small η causes a slow convergence rate, whereas a large η causes oscillations that nodes jump back and forth and thus fail to converge to useful coordinates. An additional challenge is that some nodes turn out to have a higher error in their coordinates than others. If a node communicates with some nodes that predict RTTs badly, any update based on the coordinates of those nodes is likely to increase prediction error rather than decrease it.

Vivaldi addresses these issues by varying η depending on an error measure that a node has about its coordinates. In particular, when node i updates its coordinates with respect to node j, Vivaldi adapts η at node i by

$$\eta \propto \frac{\text{error}_i}{\text{error}_i + \text{error}_j}. \tag{5.5}$$

The local error of each node indicates the confidence of the node to its coordinates. Computing the timestep in this way provides some nice properties such as quick convergence, low oscillation, and resilience against high-error nodes.

5.2.3 Harvard Vivaldi

Since proposed, Vivaldi has drawn intensive attentions of the society due to its simplicity and practicability. Numerous extensions and variations have been proposed, among which the one made by the networking group at Harvard University, introduced in [40], is particularly interesting as it addressed the issues when deploying Vivaldi in Azureus. Below, this variation of Vivaldi is called *Harvard Vivaldi*, to make the distinction.

Two issues addressed in Harvard Vivaldi are worth mentioning. First, a drift phenomenon was observed in Vivaldi, showing that the coordinates of network nodes translate and rotate as a whole in a fairly constant direction. This drift has its roots in the invariance of the Euclidean distances under a rigid transform. Thus, Harvard Vivaldi introduced a gravity term that attracts the nodes to the origin of the coordinate system. The gravity forces the mass of the nodes to be centered at the origin and successfully overcomes the translation. However, the rotation of the coordinates remains as the nodes can still rotate around the origin. Section 5.5.5 will show that DMFSGD has a similar behavior due to a similar invariance property.

Second, it was observed that in Azureus, the RTT measurements were obtained in a passive manner during data transfer. A related problem is *skewed update*. In words, some nodes may

stay in the system for much longer time than others and these nodes will be probed at greater frequency due to their longer life cycle. A direct consequence is that the energy minimization will become skewed toward these nodes, i.e. their coordinates are more accurate because they make more updates and release their energy more thoroughly. To overcome this problem, a weighting strategy is incorporated that scales the weight of a node by its age so that older information receives less weight. This strategy is also incorporated in DMFSGD, which will be introduced in Section 5.3.2.

5.2.4 Discussions

Although Vivaldi and its variations can achieve reasonably accurate results in a large variety of situations, a big issue is that network distances estimated by Vivaldi are constrained by the property of the Triangle Inequality. However, it has been observed that the Violation of the Triangle Inequality (TIV) in the network distance space is persistent and widespread. The TIV has become a barrier that prevents the further improvements of the prediction accuracy of Vivaldi. This impact will be demonstrated in Section 5.5.

5.3 Network Distance Prediction by DMFSGD

The basic DMFSGD algorithm for network distance is given in Algorithm 1 in the previous chapter. This section introduces some extensions to the basic algorithm that address the measurement issues mentioned in Section 5.1

5.3.1 Minibatch and Line Search

The basic DMFSGD algorithm is sensitive to the learning rate η. This sensitivity can be relieved by using more training samples at the same time, leading to minibatch SGD. In this context, applying minibatch SGD amounts to updating with respect to all available measurements at each node, instead of one measurement at a time.

In particular, define the losses related to all measurements available at node i as

$$l_i = \sum_{j \in \mathcal{N}_i} l(x_{ij}, u_i v_j^T) + \lambda u_i u_i^T, \tag{5.6}$$

$$l^i = \sum_{j \in \mathcal{N}_i} l(x_{ij}, u_j v_i^T) + \lambda v_i v_i^T. \tag{5.7}$$

Recall that $\mathcal{N}_i$ is the neighbor set of node i. Essentially, l_i is the regularized loss of the paths from node i to other nodes and l^i is that of the paths from other nodes to node i.

Thus, the update rules in minibatch SGD can be derived similarly as the SGD in the previous

Algorithm 3 Line Search (for updating u_i)

1: compute l_i^0 by eq. 5.6;
2: initialize η with a large value;
3: **for** $i = 1$ to $maxNumberLineSearch$ **do**
4: compute u_i by eq. 5.8;
5: compute l_i by eq. 5.6;
6: **if** $l_i < l_i^0 + \delta$ **then**
7: **return**
8: **end if**
9: $\eta \longleftarrow \eta/2$;
10: **end for**

chapter by gradient descent, leading to

$$u_i = (1 - \eta\lambda)u_i - \eta \sum_{j \in \mathcal{N}_i} \frac{\partial l(x_{ij}, u_i v_j^T)}{\partial u_i}, \tag{5.8}$$

$$v_i = (1 - \eta\lambda)v_i - \eta \sum_{j \in \mathcal{N}_i} \frac{\partial l(x_{ij}, u_j v_i^T)}{\partial v_i}, \tag{5.9}$$

To completely get rid of η, a line search strategy can be incorporated to determine η adaptively [5]. In particular, in each update, η starts with a large initial value and is gradually decreased until the losses in eqs. 5.6 or 5.7 are reduced after the update. The line search algorithm for updating u_i is given in Algorithm 3. The same algorithm can be used for updating v_i by replacing eq. 5.6 by eq. 5.7 and eq. 5.8 by eq. 5.9. Note that δ in Line 6 is a small positive constant that helps overcome poor local optimums. The effectiveness of adapting η by line search using Algorithm 3 will be demonstrated in Section 5.5.

5.3.2 Neighbor Decay and Neighbor Selection

The minibatch strategy requires each node to probe all neighbors at the same time, which is impractical. To avoid this, each node maintains the recent information (distance measurements and coordinates) of its neighbors. In minibatch SGD, each node probes one neighbor at a time but updates its coordinate with respect to all neighbors in the neighbor set using the recorded historical information.

A neighbor decay strategy is incorporated that scales the weight of each node in the neighbor set by its age so that older information receives less weight, i.e.,

$$w_j^i = \frac{a_{max} - a_j}{\sum_{j \in \mathcal{N}_i}(a_{max} - a_j)}, \tag{5.10}$$

where a_j is the age of the information of node j and a_{max} is the age of the oldest information in

the neighbor set. Thus, the weighted update rules are

$$u_i = (1 - \eta\lambda)u_i - \eta \sum_{j \in \mathcal{N}_i} w_j^i \frac{\partial l(x_{ij}, u_i v_j^T)}{\partial u_i}, \tag{5.11}$$

$$v_i = (1 - \eta\lambda)v_i - \eta \sum_{j \in \mathcal{N}_i} w_j^i \frac{\partial l(x_{ij}, u_j v_i^T)}{\partial v_i}, \tag{5.12}$$

Note that this neighbor decay strategy was firstly proposed by [40] to overcome the problem of skewed neighbor update in Vivaldi. As mentioned in Section 5.2.3, the optimization becomes skewed toward those nodes with long life cycle, because they make more updates.

Conventionally, the neighbors of a node are selected randomly and the distances between a node and its neighbors are probed by active measurements [22]. However, in practice, the distance measurements are often acquired passively with no measurement cost. As mentioned in Section 5.1, this passivity is enforced in applications such as Azureus.

Therefore, the two situations are differentiated where network distances are probed by active and passive measurements. For the former, the conventional random neighbor selection procedure is adopted, i.e., each node randomly selects k nodes as its neighbors and actively probes one of them from time to time. For the latter, no neighbor selection is performed explicitly and each node maintains a relatively small set of active neighbors with which it recently communicated and updates its coordinates whenever a new measurement is made available. Note that this difference has no impact on the update rules in eqs. 5.8 and 5.9 or in eqs. 5.11 and 5.12.

5.3.3 Robust Matrix Factorization

In the basic DMFSGD algorithms in the previous chapter, the commonly-used L_2 loss function is adopted. However, it is well known that the L_2 loss function is sensitive to large noises and outliers which often occur in network measurements due to network anomaly such as sudden traffic bursts and attacks from malicious nodes. Other loss functions such as the L_1 loss function, the ϵ-insensitive loss function and the Huber loss function are more robust and can tolerate outliers [33, 37]. For example, the L_1 loss function is defined as

$$l(x, \hat{x}) = |x - \hat{x}|. \tag{5.13}$$

Figure 5.3 shows the L_1 and L_2 loss function.

Thus, potentially, the robustness of matrix factorization can be enhanced by replacing the L_2 loss function by e.g. the L_1 loss function, and the same SGD procedure can be applied to solve the robust matrix factorization problem. Note that the L_1 loss function is non-differentiable and the gradients have to be approximated by the subgradients. Analogously, the subgradient-based technique that optimizes non-differentiable functions is called subgradient descent [5]. Following the convention in [8], the term SGD is used and referred to as both Stochastic Gradient and SubGradient Descent.

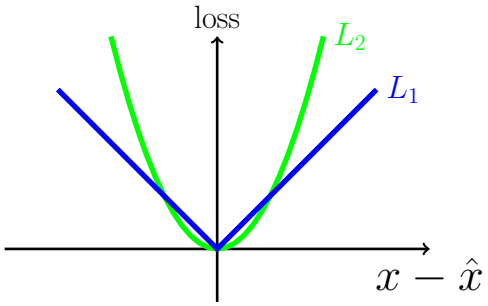


Figure 5.3: The L_1 (blue) and L_2 (green) loss function.

The gradients of the L_1 loss function in the update rules in eqs. 5.8 and 5.9 or in eqs. 5.11 and 5.12 are given below. Without causing any confusion, the subscripts are dropped.

$$\frac{\partial l(x, uv^T)}{\partial u} = -\text{sign}(x - uv^T)v, \tag{5.14}$$

$$\frac{\partial l(x, uv^T)}{\partial v} = -\text{sign}(x - uv^T)u. \tag{5.15}$$

Comparing the gradient functions of the L_2 and the L_1 loss functions in eqs. 4.8 and 4.9 and in eqs. 5.14 and 5.15, the only difference is that for the L_2 loss function, the magnitudes of the gradients are proportional to the fitting errors $(x - uv^T)$, whereas for the L_1 loss function, only the signs of the fitting errors are taken into consideration and decide the directions of the updates.

5.3.4 Nonnegative Matrix Factorization

Conventional matrix factorization techniques do not preserve the non-negativity of the distances. Empirically, only a very small portion of the predicted distances were found negative by the DMFSGD algorithm, and a direct solution is to turn $\hat{x}_{ij}$ into a small positive value if $\hat{x}_{ij} = u_i v_j^T < 0$.

A systematic solution is to incorporate the non-negativity constraint in matrix factorization, leading to the nonnegative matrix factorization (NMF) that optimizes

$$\sum_{(i,j)\in\Omega} l(x_{ij}, u_i v_j^T) + \lambda \sum_{i=1}^{n} u_i u_i^T + \lambda \sum_{i=1}^{n} v_i v_i^T, \tag{5.16}$$
$$\text{subject to } u_i \geqslant 0,\ v_i \geqslant 0,\ i = 1, \ldots, n.$$

The non-negativity constraint guarantees the entries in (u_i, v_i) are non-negative.

The optimization of NMF is not fundamentally different from that of the unconstrained matrix factorization, adding only one projection step in SGD or minibatch SGD that turns the negative entries in u_i and v_i into zero after each update which causes no noticeable impact on the speed of the algorithm. The technique is also known as projected gradient descent [47]. Note that

the non-negativity constraint has been previously studied in [52] for network distance prediction, which adopted a more heavyweight nonnegative least-squares solver.

5.3.5 Symmetric Matrix Factorization

Also note that network distances or RTTs are symmetric and that this symmetry is not preserved either. A direct solution is to turn the predicted distances symmetric by defining a symmetric distance function as

$$\hat{x}^s_{ij} = \frac{\hat{x}_{ij} + \hat{x}_{ji}}{2} = \frac{u_i v_j^T + u_j v_i^T}{2}. \tag{5.17}$$

Thus, the symmetric matrix factorization problem is to search for a low-rank symmetric matrix so that

$$\hat{X}^s = \frac{UV^T + VU^T}{2} \approx X.$$

Correspondingly, the loss function to be optimized becomes

$$\sum_{(i,j)\in\Omega} l(x_{ij}, \hat{x}^s_{ij}) + \lambda \sum_{i=1}^{n} u_i u_i^T + \lambda \sum_{i=1}^{n} v_i v_i^T. \tag{5.18}$$

The SGD update rules can be derived similarly, given by

$$u_i = (1 - \eta\lambda)u_i - \eta \sum_{j\in\mathcal{N}_i} w^i_j \frac{\partial l(x_{ij}, \hat{x}^s_{ij})}{\partial u_i}, \tag{5.19}$$

$$v_i = (1 - \eta\lambda)v_i - \eta \sum_{j\in\mathcal{N}_i} w^i_j \frac{\partial l(x_{ij}, \hat{x}^s_{ij})}{\partial v_i}, \tag{5.20}$$

5.3.6 Height Model

The height model in Vivaldi [22] can also be incorporated. This model augments the (u, v) coordinate of a node with a height h. Similarly, the (u, v) coordinate models the high-speed Internet core, while the height h models the time packets take to travel the access link from the node to the Internet core. The cause of the access link distance includes queuing delay and low bandwidth [22]. The height augmented symmetric distance is defined as

$$\hat{x}^{hs}_{ij} = \frac{u_i v_j^T + u_j v_i^T}{2} + h_i + h_j. \tag{5.21}$$

Correspondingly, the loss function to be optimized becomes

$$\sum_{(i,j)\in\Omega} l(x_{ij}, \hat{x}^{hs}_{ij}) + \lambda \sum_{i=1}^{n} u_i u_i^T + \lambda \sum_{i=1}^{n} v_i v_i^T. \tag{5.22}$$

Algorithm 4 DMFSGD_RTT_Extended(i, j)

node i retrieves x_{ij}, u_j, v_j actively or passively;
node i updates the weights of its neighbors by eq. 5.10;
if use L_2 loss function **then**
    adopt the gradients in eqs. 4.8 and 4.9;
else {use L_1 loss function}
    adopt the gradients in eqs. 5.14 and 5.15;
end if
update u_i by eq. 5.11 with η set by line search;
update v_i by eq. 5.12 with η set by line search;
if force non-negativity **then**
    turn the negative entries in u_i and v_i into 0;
end if

The SGD update rules can be derived similarly, given by

$$u_i = (1 - \eta\lambda)u_i - \eta \sum_{j \in \mathcal{N}_i} w_j^i \frac{\partial l(x_{ij}, \hat{x}_{ij}^{hs})}{\partial u_i}, \tag{5.23}$$

$$v_i = (1 - \eta\lambda)v_i - \eta \sum_{j \in \mathcal{N}_i} w_j^i \frac{\partial l(x_{ij}, \hat{x}_{ij}^{hs})}{\partial v_i}, \tag{5.24}$$

$$h_i = (1 - \eta\lambda)h_i - \eta \sum_{j \in \mathcal{N}_i} w_j^i \frac{\partial l(x_{ij}, \hat{x}_{ij}^{hs})}{\partial h_i}, \tag{5.25}$$

5.3.7 Extended DMFSGD Algorithm

The above extensions can be easily incorporated in the basic DMFSGD algorithm in Algorithm 1. Empirically, no or little improvement was found by incorporating the symmetric or height-augmented symmetric distance function in eq. 5.17 or 5.21, which is probably because both $\hat{x}_{ij}$ and $\hat{x}_{ji}$ are forced to approximate x_{ij}, as mentioned in Section 4.3.1 in Chapter 4, and because the RTT measurements in the datasets used in this chapter were collected from servers or network nodes that are close to the Internet core. While neither the symmetric matrix factorization nor the height model are included in the prediction system, the other extensions not only improve the accuracy but also make the results more stable and less sensitive to parameter settings, which will be demonstrated in Section 5.5.

The extended DMFSGD algorithm is given in Algorithm 4. Note that the extended version has the same architecture as the basic version in Algorithm 1 and Vivaldi. It employs the same process at all nodes, with update rules containing only vector operations. In the rest of the chapter, without causing any confusion, the extended version in Algorithm 4 is simply referred to as DMFSGD.

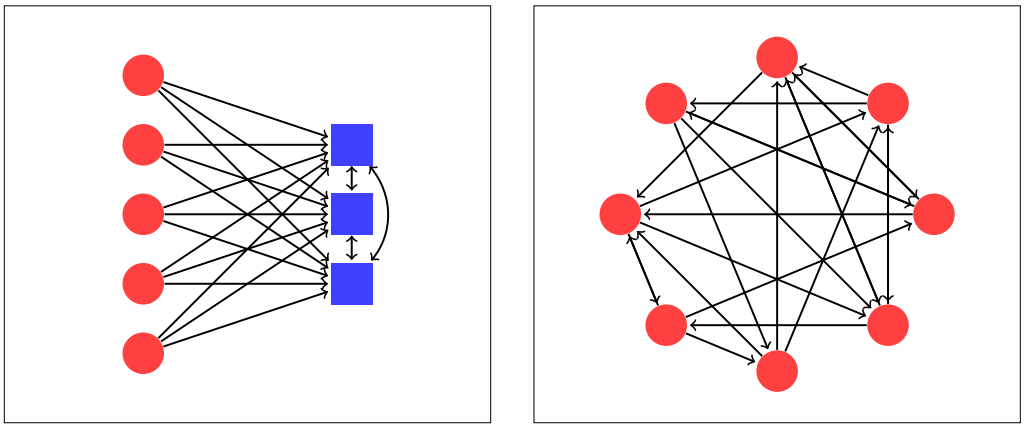


Figure 5.4: Architectures of landmark-based, the left plot, and decentralized, the right plot, systems for network distance prediction. The squares are landmarks and the circles are ordinary nodes. The directed path from node i to node j means that node i probes node j and therefore $(i, j) \in \Omega$.

5.4 A Unified View of Network Distance Prediction

This section provides a unified view of different approaches to network distance prediction.

5.4.1 A Unified Formulation

Numerous approaches to network distance prediction have been proposed, including GNP [58], Vivaldi [22], IDES [52], DMFSGD, and many variations of these approaches, which have adopted different models, based on Euclidean embedding or matrix factorization, and different architectures, either with landmarks or without. Nevertheless, these seemingly different approaches all optimize an objective function of the following form:

$$L(X, \hat{X}, \Omega) = \sum_{(i,j)\in\Omega} l(x_{ij}, \hat{x}_{ij}). \tag{5.26}$$

They differ only in the sampling of the paths to be measured in Ω, or equivalently the selection of neighbors at each node, and in the associated distance function to calculate $\hat{x}_{ij}$.

Sampling of Paths

For landmark-based methods, as all paths between landmarks are measured and ordinary nodes probe only the landmarks, thus a path is sampled if the target node is a landmark, i.e.,

$$(i, j) \in \Omega, \text{ if } j \text{ is a landmark.}$$

For decentralized methods that rely on no landmark, as each node equally probes a number of neighbors, the measured paths in Ω are sampled either actively and randomly or passively with no control. Figure 5.4 illustrates the architectures of landmark-based and decentralized systems.

Definition of Distance Function

For matrix factorization, recall that the predicted distances are calculated by a dot product between node coordinates, given in eq. 4.2. For Euclidean embedding, the Euclidean distance function is used and given in eq. 5.1.

The different choices of the distance function naturally impose different constraints on the produced results. For instance, the use of the Euclidean distance in eq. 5.1 forces $\hat{X}$ to satisfy the geometric constraints of the symmetry and the triangle inequality, while the use of the dot-product distance in eq. 4.2 forces $\hat{X}$ to be low-rank. Table 5.1 compares the main features of matrix factorization and Euclidean embedding.

Table 5.1: Matrix Factorization vs. Euclidean Embedding

	Matrix Factorization	Euclidean Embedding
optimization objective	$\sum_{(i,j)\in\Omega} l(x_{ij}, \hat{x}_{ij})$	$\sum_{(i,j)\in\Omega} l(x_{ij}, \hat{x}_{ij})$
distance function	$\hat{x}_{ij} = u_i v_j^T$	$\hat{x}_{ij} = \sqrt{(u_i - u_j)^T (u_i - u_j)}$
node coordinate	$u_i = (u_{i1}, \cdots, u_{ir})$ $v_i = (v_{i1}, \cdots, v_{ir})$	$u_i = (u_{i1}, \cdots, u_{ir})$
constraints	$\hat{X}$ is low rank	Symmetry: $\hat{x}_{ij} = \hat{x}_{ji}$ Triangle Inequality: $\hat{x}_{ij} < \hat{x}_{ik} + \hat{x}_{kj}$

5.4.2 A Unified Framework

The unified formulation suggests that at the heart of network distance prediction is a common optimization problem which can in principle be solved by any optimization scheme. This insight enables a unified framework to solve the problem under different models and architectures. For instance, the SGD-based framework of the DMFSGD algorithm is generic and can also deal with the landmark-based architecture, by letting each node only select landmarks as its neighbors, and with Euclidean embedding, by substituting the dot-product distance in eq. 4.2 by the Euclidean distance in eq. 5.1, which leads essentially to the update rule of Vivaldi, given by

$$u_i = u_i - \eta \frac{\partial l(x_{ij}, \hat{x}_{ij})}{\partial u_i} = u_i + \eta (x_{ij} - \hat{x}_{ij}) \frac{u_i - u_j}{\hat{x}_{ij}}. \tag{5.27}$$

Recall that u_i is the Euclidean coordinate of node i.

As mentioned in Section 5.2, Vivaldi adopted the L_2 loss function with no regularization incorporated, and the learning rate η, termed differently as *timestep*, was adapted by taking into account some confidence measure of each node to its coordinate. Thus, Vivaldi can be viewed as a SGD-based decentralized Euclidean embedding algorithm, instead of the simulation of a spring system in [22].

5.5 Experiments and Evaluations

In this section, the DMFSGD algorithm is evaluated and compared with the state-of-the-art approach, Vivaldi.

5.5.1 Evaluation Methodology

Evaluation Criteria

The evaluations were performed under the following criteria.

- **Cumulative Distribution of Relative Estimation Error** Relative Estimation Error (REE) is defined as
$$REE = \frac{|\hat{x}_{ij} - x_{ij}|}{x_{ij}}.$$
- **Stress** Stress measures the overall fitness and is used to illustrate the convergence of the algorithm, defined as
$$stress = \sqrt{\frac{\sum_{i,j=1}^{n} (x_{ij} - \hat{x}_{ij})^2}{\sum_{i,j=1}^{n} {x_{ij}}^2}}.$$
- **Median Absolute Error** Median Absolute Error (MAE) is defined as
$$MAE = median_{ij}(|x_{ij} - \hat{x}_{ij}|).$$

Datasets

The evaluations were performed on the following datasets.

- **Harvard** contains dynamic and passive measurements of application-level RTTs, with timestamps, between 226 Azureus clients collected in 4 hours [40].
- **P2PSim** was obtained from the P2PSim project that contains static RTT measurements between 1740 Internet DNS servers [30].
- **Meridian** was obtained from the Cornell Meridian project that contains static RTT measurements between 2500 nodes [87].
- **P2PSim-complete** is a complete submatrix between 525 nodes derived from P2PSim.
- **Meridian-complete** is a complete submatrix between 2255 nodes derived from Meridian.
- **Synthetic-complete** contains the pairwise distances between 1000 nodes that are randomly generated in a 10-dimensional Euclidean space.

The first five datasets were obtained from real-world networks and contain a large percentage of TIV edges, whereas the last one was synthesized and is TIV-free. Here, an edge $\overline{AB}$ is claimed to be a TIV if there exists a triangle $\triangle ABC$ where $\overline{AB} > \overline{BC} + \overline{AC}$. The last three datasets are only used in section 5.5.2 for the purpose of comparing the models of Euclidean embedding and matrix factorization.

Table 5.2 summarizes these datasets. Note that it is impossible to tell the symmetry or to calculate the TIV percentage of the Harvard dataset, as the measurements between network nodes vary over time largely, sometimes in several orders of magnitudes. The Harvard dataset is rather dense with about 3.9% pairwise paths unmeasured in 4 hours. The other paths are measured in uneven frequencies with one measured the maximal 662 times and one the minimal 2 times. About 94.0% of the paths are measured between 40 and 60 times.

Table 5.2: Properties of The Datasets

Dataset	Nodes	Symmetry	TIV percentage	Dynamic
Harvard	226	/	/	Yes
P2PSim	1740	Yes	85.53%	No
Meridian	2500	Yes	96.55%	No
P2PSim-complete	525	Yes	76.17%	No
Meridian-complete	2255	Yes	96.25%	No
Synthetic-complete	1000	Yes	0	No

Implementations for Different Datasets

As mentioned in Section 5.3.2, the DMFSGD algorithm adopts the conventional random neighbor selection procedure in the scenarios where measurements are probed actively and maintains dynamically an active neighbor set for each node in the scenarios where measurements are obtained passively. Thus, for the Harvard dataset, each node maintains an active neighbor set containing the nodes it has contacted within the past 30 minutes and the timestamped measurements are processed in time order. For the other datasets, the random neighbor selection is used and the measurements are processed in random order with no neighbor decay (Line 2 in Algorithm 4) as they are static.

To handle the dynamics of the measurements in Harvard, the distance filter in [40] is adopted that smooths the streams of measurements within a moving time window, 30 minutes in this paper, by a median filter. In the evaluation, a static distance matrix was constructed by extracting the median values of the streams of measurements between each pair of nodes and was used as the ground truth.

5.5.2 Euclidean Embedding vs. Matrix Factorization

Firstly, the models of Euclidean embedding and matrix factorization are compared to highlight the suitability of the low-rank constraint over the Euclidean distance properties on the modeling of the network delay space.

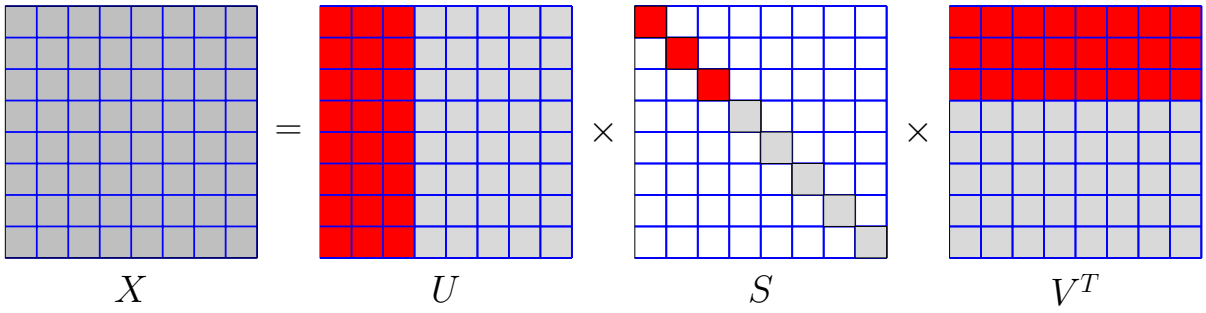


Figure 5.5: Singular value decomposition. In the example, X has 3 non-zero singular values.

Algorithms

As the goal is to compare the model suitability, ideally we should choose the algorithms that find the global optimum for both models and check which optimum is better. For matrix factorization, SVD provides the analytic and globally optimal solution for a complete matrix [28]. Generally, SVD factorizes X into three matrices, i.e.,

$$X = USV^T,$$

where U and V are unitary matrices, and S is a diagonal matrix with nonnegative real numbers on the diagonal. The positive diagonal entries are called the singular values and their number is equal to the rank of X. Figure 5.5 illustrates the process of singular value decomposition.

To obtain a low-rank factorization, only the r larger singular values in S are kept and the other small ones are replaced by zero. Let S_r be the new S, $U_r = US_r^{\frac{1}{2}}$ and $V_r^T = S_r^{\frac{1}{2}}V^T$, where $S_r(i,i)^{\frac{1}{2}} = \sqrt{S_r(i,i)}$. Then, $\hat{X} = U_r V_r^T$ is the globally optimal low-rank approximation to X.

For Euclidean embedding, also known as MultiDimentional Scaling (MDS), no algorithm can find a global optimum because this problem is non-convex [7]. Nevertheless, the MDS implementation in matlab, mdscale [1], has been widely used and was found to work well on the given datasets. To reduce the chance of being trapped in a poor local optimum, mdscale was run for multiple times with different random initializations and similar solutions were found in every run, which gives a good confidence in the results achieved.

Experiments and Observations

The model comparison was performed on the three complete datasets including Synthetic-complete, P2PSim-complete and Meridian-complete. On each dataset, MDS and SVD were run in different dimensions and ranks and we computed the stresses and MAE, shown in figure 5.6. It can be seen that the accuracies by SVD monotonically improve on all three datasets as the rank increases, whereas consistent improvements by MDS are only found on Synthetic-complete which is TIV-free. On P2PSim-complete and Meridian-complete where severe TIVs exist, MDS achieves no or little visible improvement beyond 10 dimensions.

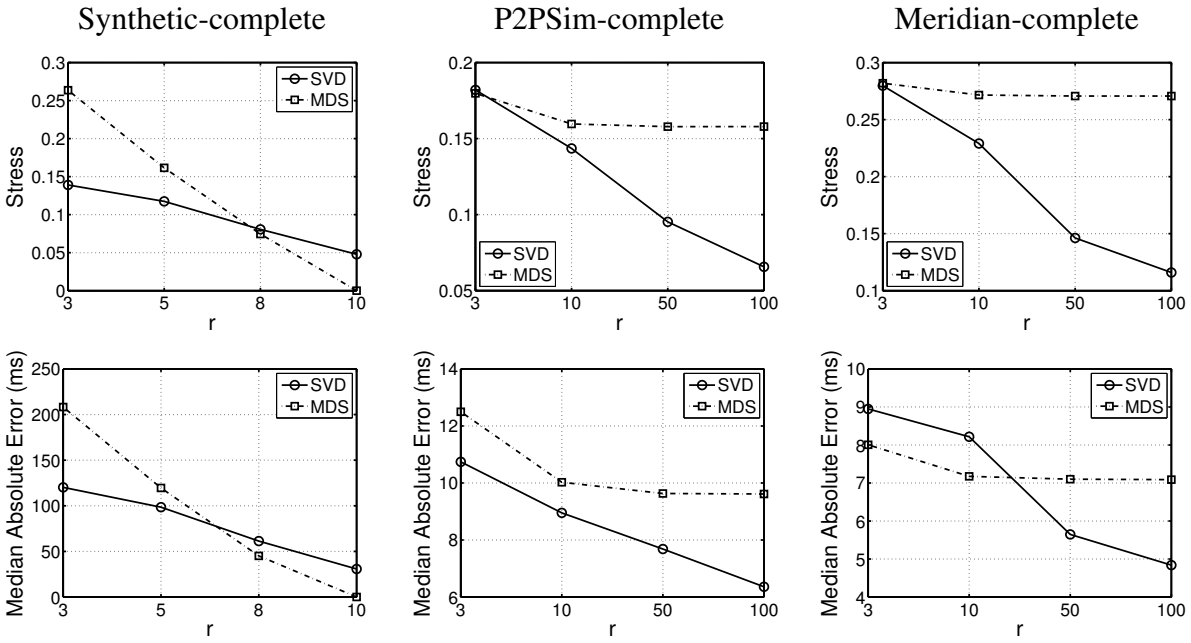


Figure 5.6: Comparison of MDS-based Euclidean embedding and SVD-based matrix factorization on Synthetic-complete, P2PSim-complete and Meridian-complete. The stresses and the median absolute errors by both methods in different dimensions/ranks are shown on the first and second rows respectively. Note that a perfect embedding with no errors was generated for Synthetic-complete in the 10 dimensional Euclidean space by MDS.

These evaluations demonstrate the influences of different constraints imposed on the two models. For Euclidean embedding, the symmetry constraint does not cause any problem as the RTTs in all datasets are symmetric. However, the constraint of triangle inequality is strong and cannot be relieved by increasing dimensions. In contrast, matrix factorization makes no assumptions of triangle inequality, thus is not affected by the TIVs in the data. Note that for matrix factorization, the accuracy improvement by increasing the rank is guaranteed when the matrix is complete. Section 5.5.3 will show that in the presence of a large amount of missing data, increasing the rank beyond some value will not further improve the accuracy.

This comparative study reveals the model advantage of matrix factorization over Euclidean embedding. Overall, Euclidean embedding has a geometric interpretation which is useful for visualization. However, due to the existence of TIVs and the possible asymmetry of network delays, the low-rank constraint in matrix factorization is more suitable for modeling the network delay space on the Internet. Section 5.5.4 will show that this advantage takes effect when dealing with real incomplete network distance matrices using the DMFSGD algorithm.

5.5.3 Impact of Parameters

This section discusses and demonstrates the impact of the parameters of the DMFSGD algorithm.

Number of Neighbors k

In the mode of active probing of measurements, k controls the amount of data that is known to each node. Thus, increasing k always helps improve accuracies as more data becomes available. However, a larger k also means more probing traffic and consequently higher overheads. Thus, k has to be chosen by trading off between accuracies and overheads. Following the suggestion in Vivaldi [22], $k = 32$ is set by default for P2PSim and Meridian. Note that $k = 32$ makes the available measurements considerably sparse. For instance, $32/1740 = 1.84\%$ measurements are available for each node in P2PSim and $32/2500 = 1.28\%$ for each node in Meridian. Recall that no k is set for Harvard.

Rank r

Given a delay matrix X, its rank r depends on a number of network properties such as the node distribution, the network topologies and the routing policies. On the one hand, r should be large enough so that no significant singular values of X are abandoned. On the other hand, recovering X with a larger r demands more data, increasing measurement overheads. Thus, an interesting question is, given a certain number of measurements, what is the proper rank leading to an accurate recovery? Below, this question is answered empirically for the given datasets.

Regularization λ

λ controls the overfitting and improves the numerical stability of the solutions.

Learning Rate η

As mentioned in Section 5.3.1, SGD is sensitive to η where a too large η leads to the failure of convergence and a too small η leads to the slow convergence. Thus, η is adapted by the line search, with the initial value of 10^{-3} for the L_2 loss function and of 10^{-2} for the L_1 loss function.

Experiments and Observations

In the first experiment, different configurations were tested and compared, with $r = \{3, 10, 100\}$ and $\lambda = \{0.01, 0.1, 1, 10\}$ and with different loss functions and whether to incorporate the nonnegativity constraint, shown in Figure 5.7. In this experiment, η is adapted by the line search.

In particular, the following observations were made. First, the DMFSGD algorithm is generally more accurate when the robust L_1 loss function and the nonnegativity constraint are incorporated. The likely reasons are that the L_1 loss function is insensitive to large fitting errors, some of which are introduced by measurement noises and outliers, and that the nonnegativity constraint reduces the searching space which makes it easier to find a stable solution. Thus, the L_1 loss function and the nonnegativity constraint are incorporated in the DMFSGD algorithm by default.

Second, $\lambda = 1$ seems to be a good choice under most configurations and is thus adopted by default. Third, the impact of r depends on the properties of the dataset. In Harvard where available measurements are dense, the prediction accuracy improves monotonically with r, whereas in the other two datasets where available measurements are sparse due to the setting of a small k, better performance is achieved with $r \leqslant 10$. This observation suggests that r is closely related to the availability of data. For instance, a certain k allows the accurate recovery for only a certain r. Increasing r beyond some value for a given k will not improve the accuracy but only cause severe overfitting. Thus, by trading off between the performance on all three datasets and by taking into account that available measurements are often limited, a relatively small value of $r = 10$ is adopted by default.

In the second experiment, different configurations of the learning rate η were tested and compared including a few constant η's and the line search that adapts η dynamically. Results are shown in Figure 5.8. It can be seen that the line search strategy performs best in terms of both accuracy and convergence speed. Note that the convergence speed is illustrated by the stress and MAE improvements with respect to the average measurement number per node, i.e. the total number of measurements used by all nodes divided by the number of nodes[1]. It can be seen that the DMFSGD algorithm converges fast after each node has probed, on average, $10 \times k$ measurements from its k neighbors. Although no k is set for Harvard, k is treated as 226.

Discussions

The default configuration of $\lambda = 1$ and $r = 10$ with the incorporation of the line search strategy, the L_1 loss function and the nonnegativity constraint is not guaranteed to be optimal in different situations and on different datasets. However, fine tuning of parameters is difficult, if not impossible, for network applications due to the measurement dynamics and the decentralized processing where local measurements are processed locally at each node with no central node gathering information of the entire network. Empirically, the default parameter setting leads to good, though not the best, prediction accuracy to a large variety of data.

In the mode of active probing of measurements, the number of neighbors k has to be scaled, according to the theory of matrix completion [14, 13, 38], with the number of network nodes n at least by $O(r\log n)$ to guarantee a decent accuracy. Its exact value is however data dependent, subject to the accuracy and overhead constraints, and can only be empirically determined. Thus, an experiment of different ks on P2PSim and Meridian was carried out. For each k, DMFSGD was run for 10 times, with different random neighbor selections and with different random coordinate initializations, and we calculated the mean and the standard deviation of the stress of the 10 runs, shown in Table 5.3. It can be seen that while the accuracy improves monotonically with the increase of k, the improvement becomes less significant when k goes from 32 to 64. This suggests that the choice of $k = 32$ for n as large as 2500 is indeed a good trade-off between accuracies and overheads. The small standard deviations show that DMFSGD is insensitive to both the random selection of neighbors and the random initialization of coordinates. Note that

[1]For P2PSim and Meridian, at any time, the number of measurements used by each node is statistically the same for all nodes due to the random selections of the source and the target nodes in the updates. For Harvard, this number is significantly different for different nodes because the paths were passively probed with uneven frequencies.

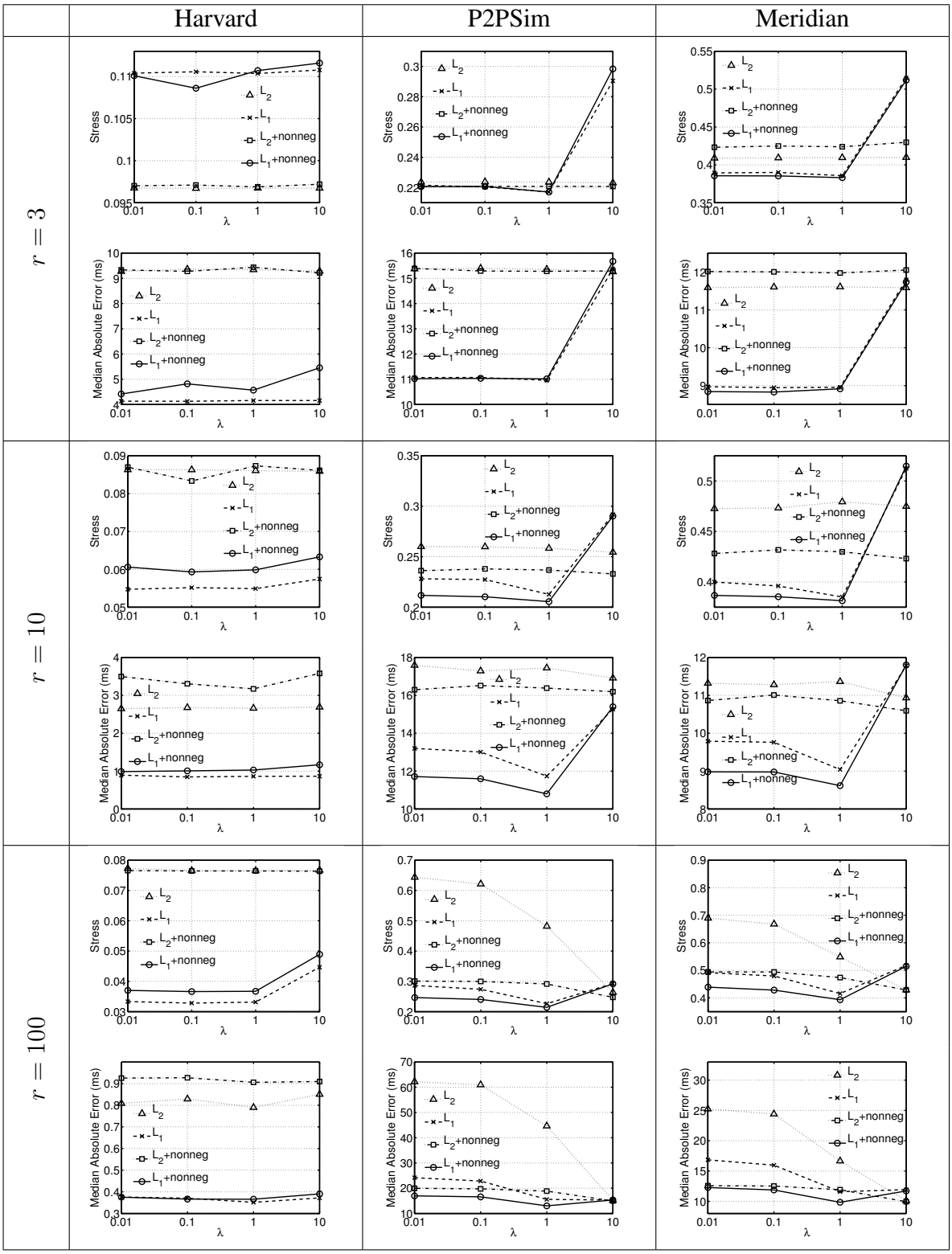


Figure 5.7: Impact of parameters. η is adapted by the line search.

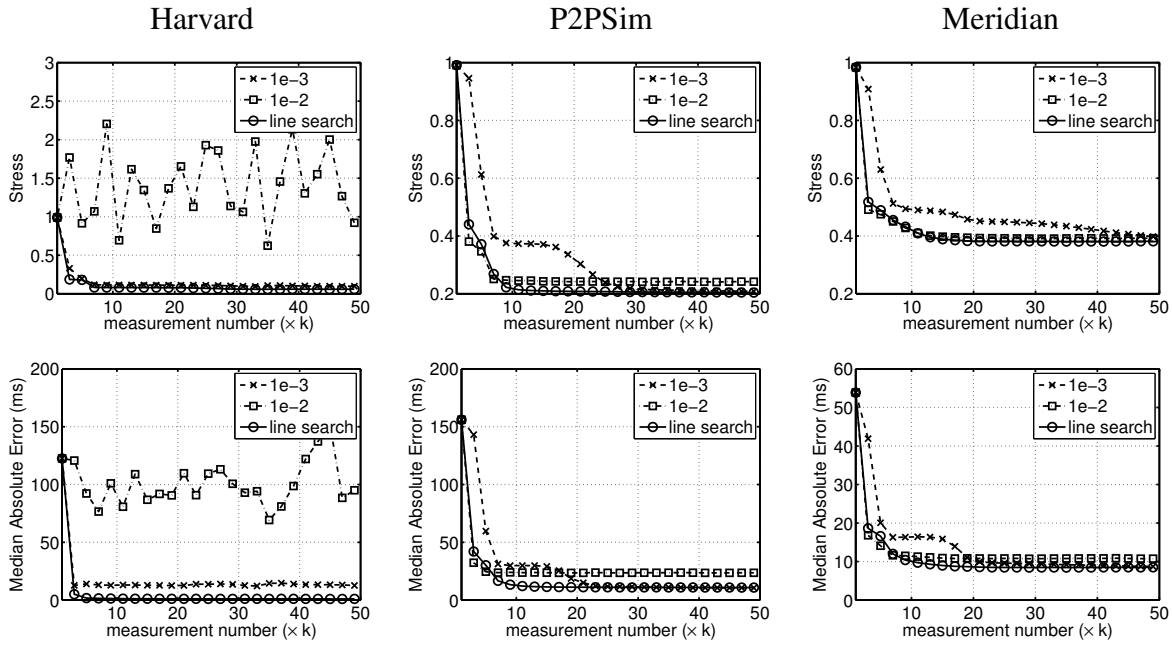


Figure 5.8: Impact of η. k is treated as 226 for Harvard and $k = 32$ for P2PSim and Meridian.

for Harvard, the 10 runs of DMFSGD were carried out with only different random coordinate initializations, with no neighbor selection.

Table 5.3: Mean and Standard Deviation of Stress

	stress	$k = 8$	$k = 16$	$k = 32$	$k = 64$
P2PSim	mean	0.3405	0.2403	0.2068	0.1871
	std	0.0064	0.0024	0.0015	0.0009
Meridian	mean	0.5048	0.4296	0.3805	0.3588
	std	0.0069	0.0038	0.0016	0.0018
Harvard	mean	0.0599			
	std	0.0010			

5.5.4 Comparisons with Vivaldi

Among numerous approaches to network distance prediction, Vivaldi [22] is widely considered as the state of the art for network distance prediction. Other approaches such as GNP [58] and IDES [52] are less convenient due to the usage of landmarks, which makes their application impossible in the context of passive probing of measurements (thus impossible to be evaluated on

the Harvard dataset). As mentioned in Section 5.4, these landmark-based systems are considered as a special variation of a generic decentralized model.

Thus, the DMFSGD algorithm is only compared to Vivaldi. To address the measurement dynamics and the skewed neighbor updates, the implementation of Harvard Vivaldi in [40] [2] was adopted when dealing with the Harvard dataset. The conventional Vivaldi in [22] was adopted to deal with the other two datasets. In addition, the flexibility of DMFSGD in dealing with the landmark-based architecture, referred to as DMFSGD Landmark, is demonstrated, despite the impracticality, by forcing each node to only select the landmarks as neighbors. Note that DMFSGD Landmark was only run on P2PSim and Meridian. To make the comparison fair, 32 landmarks were randomly selected.

Figure 5.9 shows the comparisons between DMFSGD, Vivaldi/Harvard Vivaldi and DMFSGD Landmark. It can be seen that while DMFSGD and DMFSGD Landmark perform similarly, DMFSGD either outperforms or is competitive with Vivaldi on different criteria. On Harvard, DMFSGD is significantly better on all criteria. Especially, DMFSGD achieved the $1ms$ MAE, in contrast to the $6ms$ by Harvard Vivaldi, meaning that half of the estimated distances have an error of less than $1ms$. On P2PSim and Meridian, while the stress of DMFSGD is similar to Vivaldi's, moderate improvements by DMFSGD were achieved on either the MAE or the cumulative distributions of REE. Note that among the three criteria, the MAE and the cumulative distribution of REE are considered more important as they show the accuracies of the estimates of a number of paths. The stress happens to be sensitive to the large errors in a few estimates.

The superiority of DMFSGD on Harvard is interesting. On the one hand, it demonstrates the usability of DMFSGD as Harvard contains real dynamic data collected from Azureus. On the other hand, it seems to show that DMFSGD is more advantageous in situations where dense data is available. To verify this, another experiment was performed on P2PSim and Meridian with $k = 128$, shown in Figure 5.10. It can be seen that the improvement of DMFSGD over Vivaldi is indeed more visible and obvious. This suggests that matrix factorization captures the correlations between matrix entries which can be better learned from more data. In contrast, the accuracy of Vivaldi suffers from severe TIVs in the measurements, and this model shortcoming cannot be relieved by e.g. adding more data or increasing dimensions.

The experiments in Figure 5.9 and 5.10 show that the model advantage of matrix factorization over Euclidean embedding takes more effects in situations where relatively dense data is available. Such situations arise in practice when measurements are probed passively as in the Harvard dataset. In addition, another observation is that while DMFSGD converges fast, Vivaldi appears to converge slightly faster, especially in Figure 5.9 where $k = 32$.

5.5.5 Drift of DMFSGD Coordinates

As mentioned in Section 5.2.3, [40] observed a drift phenomenon in Vivaldi that the coordinates of network nodes translate and rotate as a whole due to the invariance of the Euclidean distance under a rigid transform. Harvard Vivaldi overcame the translation by adding a gravity term, leaving the rotation remain.

[2] The source code was downloaded from `http://www.eecs.harvard.edu/~syrah/nc/`.

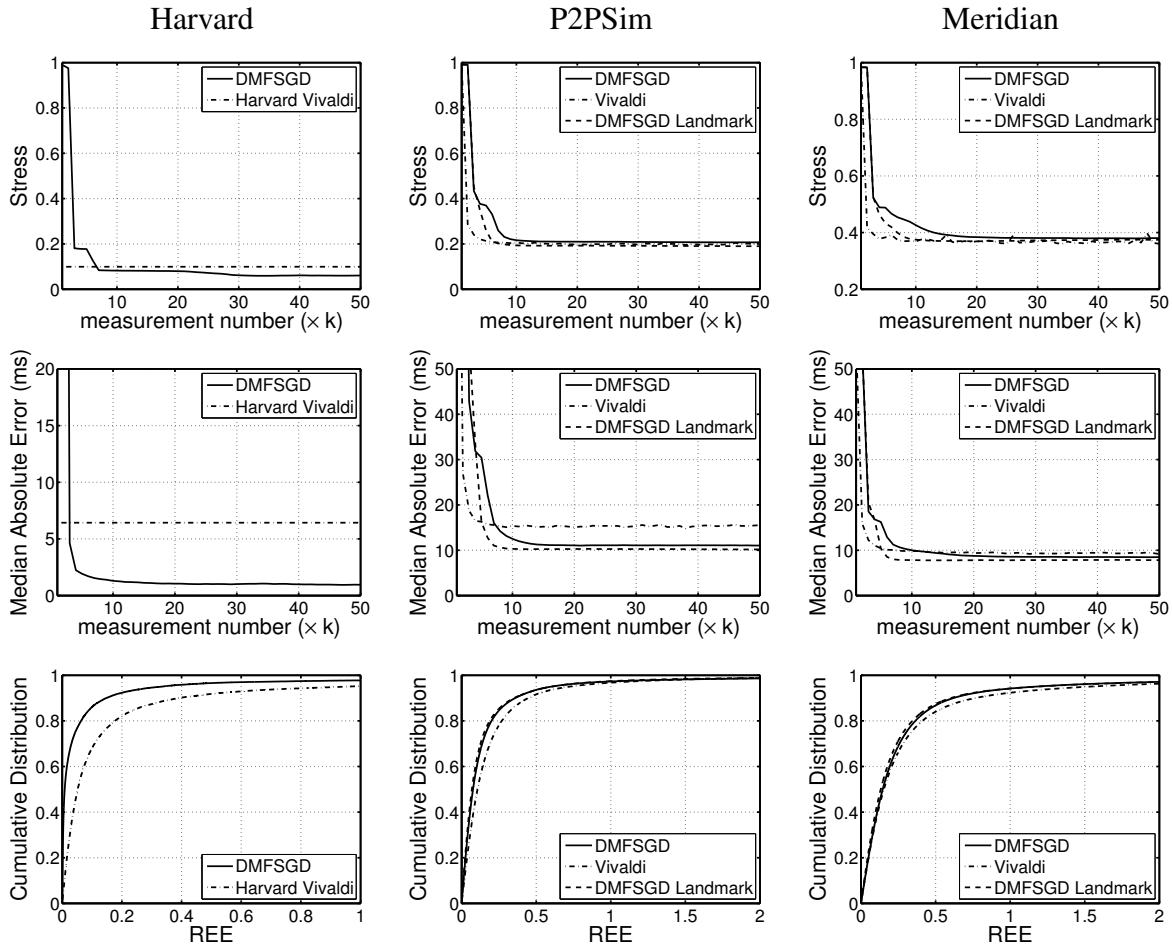


Figure 5.9: Comparison of DMFSGD and Vivaldi under $k = 32$. Note that as the implementation of Harvard Vivaldi only outputs the results in the end of the simulation, the final stress and the final MAE are plotted as a constant.

A similar behavior by DMFSGD is expected due to a similar invariance property described in eq. 3.8. The use of regularization improves the stability of the factorization by favoring solutions with lower norms. However, the factorization is still invariant under an orthogonal transform, i.e.,

$$\hat{X} = UV^T = (UR)(VR)^T, \tag{5.28}$$

where R is any arbitrary orthogonal matrix with $RR^T = I$. Thus, the pair of UR and VR are equivalent to the pair of U and V in the sense that they not only produce the same $\hat{X}$ but also have the same norm.

To verify this, we performed a simulation of a rank-3 factorization by DMFSGD for a long run and observed rotations of the coordinates of DMFSGD similar to Vivaldi's. Although such rotations do not degrade the accuracy of the predictions, their impacts on specific applications using these coordinates could be further studied. Note that an error elimination model was proposed in [84] that stabilizes the Vivaldi coordinates by progressively eliminating the prediction

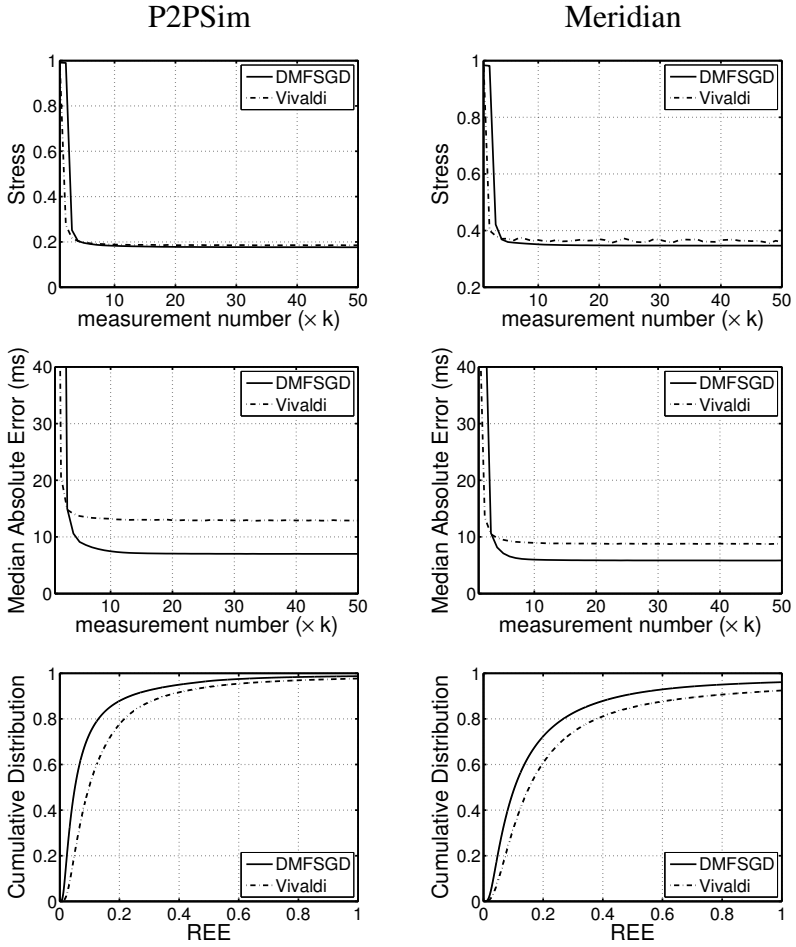


Figure 5.10: Comparison of DMFSGD and Vivaldi under $k = 128$.

errors of network paths when the errors cannot be further reduced. While this strategy could be seamlessly incorporated in DMFSGD, its effectiveness has to be verified using real application data.

5.6 Conclusions and Discussions

This chapter presents an extended DMFSGD algorithm for network distance prediction. The extensions include the minibatch and line search strategy which gets rid of the sensitive learning rate parameter, the robust matrix factorization which deals with measurement outliers, and the non-negative matrix factorization that preserves the non-negativity of the distance. The extensive experiments demonstrate the advantages of the extended DMFSGD algorithm to network distance prediction. In particular, it is justified that DMFSGD is not only accurate, often superior to and at least competitive with Vivaldi, but also practical, able to deal with dynamic measurements

in large-scale networks. These features enable an easy deployment of the prediction system in real Internet applications such as Azureus.

The extensions in DMFSGD for RTT in Algorithm 4 are also applicable to DMFSGD for available bandwidth (ABW) in Algorithm 2, with the update rules easily derived based on the same principle. However, a particular issue in the measurement of ABW is its high cost. Thus, in the next chapter, special treatments are incorporated in DMFSGD to deal with this and some other related measurement issues.

Chapter 6

Predicting End-to-End Network Performance Classes

This chapter introduces the binary classification of end-to-end network performance and the prediction of performance classes by DMFSGD. The binary classification has some nice properties such as being informative for applications, low measurement and storage cost, and unifying various metrics. The low measurement cost makes the binary classification particularly appealing for bandwidth metrics. The prediction of binary performance classes can be done by the same matrix completion framework and the same DMFSGD approach described in Chapter 4.

6.1 Binary Classification of Network Performance

Ultimately, the performance of a network path should be judged by the Quality of Service (QoS) perceived by end users. However, in practice, it is important to define an objective metric that is directly measurable without user interventions. Such metrics include network delay (RTT), available bandwidth (ABW) and many others [21], and end-to-end network performance is then quantified by the real value of a chosen metric, for example, 100ms for RTT or 15Mbps for ABW.

While such representation has been widely accepted and considered necessary for quantitative analysis in all disciplines, the exact value of a metric is hardly interesting for end users. For example, streaming media cares more about whether the ABW of a path is high enough to provide smooth playback quality. In peer-to-peer applications, although finding the nearest nodes to communicate with is preferable, it is often enough to access nearby nodes with limited loss compared to the nearest one.

Thus, in this thesis, new representations that are more qualitative than quantitative are studied. This chapter introduces the binary classification that classifies network performance into binary classes of "good" and "bad"[1], represented by 1 and -1 respectively. Such class-based representation has the following advantages over the direct representation of a metric value.

[1]Depending on the context, the class labels of "good" and "bad" may refer to "well-performing" and "poorly-performing" or "well-connected" and "poorly-connected".

- The class information already fulfills the requirements of many Internet applications. For example, in the task of intelligent path selection, the objective is generally to find a "good-enough" path instead of the optimal one. Such objective can be well served using the class information.
- Performance classes are coarse measures that are cheaper to obtain. They are also stable and better reflect long-term characteristics of network paths, which means that they can be probed less often.
- Binary classification enables the performance information to be encoded in only 1 bit, saving storage and transmission costs. In addition, it also unifies various metrics and eases their processing in applications.
- Furthermore, as the measurement of performance classes can be nicely integrated in the matrix completion framework described in Section 4.1 in Chapter 4, the same DMFSGD approach is applicable for the prediction of performance classes, with little modification required.

6.2 Measurement of Performance Classes

The measurement of end-to-end network performance is to determine a quantity of some metric. Instead of the exact value, the quantity to be determined here is a class label of "good" and "bad", represented by 1 and -1 respectively, independently from the actual metric used.

6.2.1 Classification by Thresholding

A straightforward approach to classifying network performance is by thresholding, i.e., comparing the value of the metric with a classification threshold, denoted by τ, which is determined according to the requirements of the applications. For example, a skype user may consider the performance of a network path as "good" if the delay is smaller than a tolerable bound such as 400ms [31]. Thus, $\tau = 400$ms can be defined for RTT to separate "good" paths from "bad" ones. Another example is that Google TV requires a broadband speed of 2.5Mbps or more for streaming movies and 10Mbps for High Definition contents [29]. Accordingly, $\tau = 2.5$Mbps or 10Mbps can be defined for ABW. The impact of the classification threshold τ will be discussed in Section 6.4.

Clearly, measuring network performance classes is much cheaper than measuring the exact metric value, as it only requires to determine if the metric value is larger or smaller than τ. This holds for most, if not all, metrics, since data acquisition generally undergoes the accuracy-versus-cost dilemma that accuracy always comes at a cost.

6.2.2 Measurement of ABW Classes

Measuring performance classes is particularly interesting for ABW whose measurement is costly. For example, popular tools such as pathload [34] and pathchirp [68] are based on the principle of self-induced congestion which sends large traffic at various rates so that the path being probed

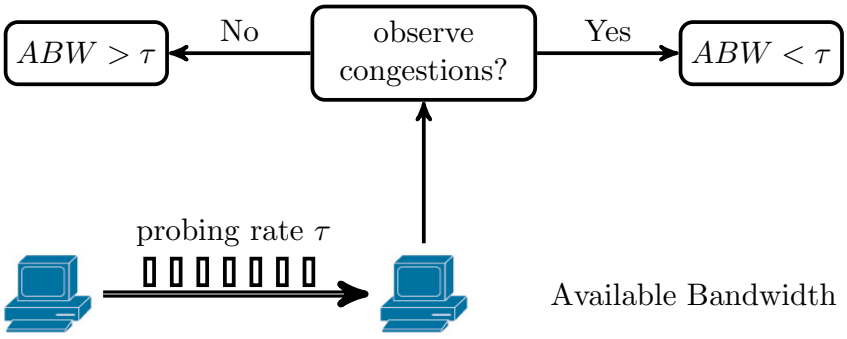


Figure 6.1: The principle of self-induced congestion for measuring ABW. Each probe by sending a constant-rate flow naturally yields a binary response of "yes" or "no", suggesting whether the ABW is larger or smaller than the probing rate.

becomes congested. The ABW is estimated as the minimum rate that creates the congestion. The difference is that pathload sends UDP trains at a constant rate and adjusts the rate from train to train, whereas pathchirp varies the probe rate within a train exponentially. A more detailed introduction can be found in Section 1.2 in Chapter 1.

The principle of self-induced congestion enables the direct measurement of ABW classes by existing tools such as pathload and pathchirp with little modification. For example, in pathload, each probe by sending a UDP train at a constant rate naturally yields a binary response of "yes" or "no" that suggests whether the ABW is larger or smaller than the probe rate, illustrated in Figure 6.1. Thus, the binary classification of ABW can be done in pathload, with only $O(1)$ probes, by setting the probe rate as the classification threshold τ and classifying the path as "good" if no congestion is observed and "bad" otherwise. This is clearly much cheaper than measuring the ABW value, because it is unnecessary to vary the probing rate in order to create congestions.

The directly measured performance classes may be inaccurate especially for those paths with metric values close to τ. The impact of the inaccuracy of the class measurements will be discussed in Section 6.4.3.

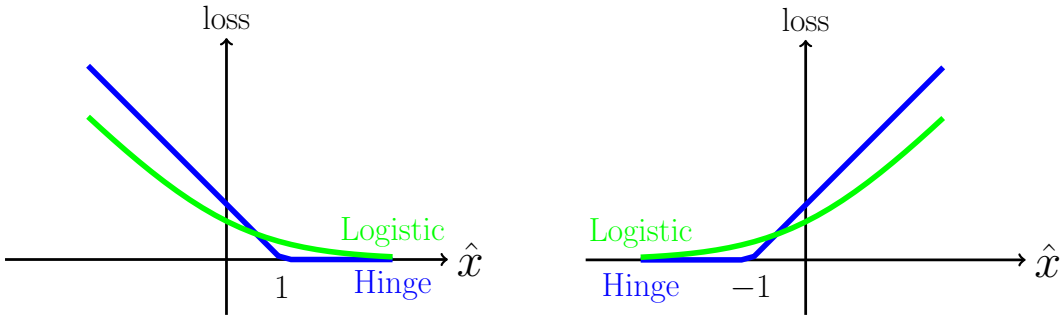
6.3 DMFSGD for Predicting Network Performance Classes

6.3.1 Formulation as Matrix Completion

The class prediction problem has the same matrix completion formulation as the one in network distance prediction in Chapter 5. Recall that the objective function is given in eq. 4.1 in Chapter 4, which is

$$\{(u_i, v_i), i = 1, \ldots, n\} = \arg\min \sum_{(i,j)\in\Omega} l(x_{ij}, u_i v_j^T) + \lambda \sum_{i=1}^{n} u_i u_i^T + \lambda \sum_{i=1}^{n} v_i v_i^T.$$

There are three main differences between the class prediction in this chapter and the network distance prediction in Chapter 5. First, the performance measure x_{ij} takes a discrete value of



(a) $x = 1$ (b) $x = -1$

Figure 6.2: The hinge (blue) and the logistic (green) loss function. In these loss functions, x is the true class label and takes a discrete value of either 1 or -1.

either 1 or -1, instead of a real-valued quantity. Second, the predicted measure $\hat{x}_{ij}$ is real-valued and has to be converted to a class by e.g. taking the sign of $\hat{x}_{ij}$. Last, the loss function l is one of the classification loss functions such as the hinge and the logistic loss function [6], given by

$$\text{hinge loss function:} \qquad l(x, \hat{x}) = \max(0, 1 - x\hat{x}); \tag{6.1}$$

$$\text{logistic loss function:} \qquad l(x, \hat{x}) = \ln(1 + e^{-x\hat{x}}). \tag{6.2}$$

Note that the hinge loss function is not differentiable. Figure 6.2 shows the two loss functions. Both the hinge and logistic loss functions are such that values of $x\hat{x}$ lower than 1 are strongly penalized and otherwise less or not penalized. These classification loss functions are thus not sensitive to the actual value of $\hat{x}$ as long as its sign matches the sign of x. For this reason, they are more suitable for classification tasks such as the classification of network performance here than the L_2 or L_1 loss function used for network distance prediction, shown in Figure 5.3.

As mentioned earlier in Section 4.1.1 in Chapter 4, in order for matrix completion to be possible, the matrix to be completed has to be low rank, exactly or approximately. To show that the low-rank assumption holds for matrices of performance classes, Figure 6.3 plots the singular values of a RTT and a ABW matrix and of their binary class matrices. It can be seen that the singular values of all matrices decrease fast, indicating a strong low-rank characteristic.

6.3.2 System Architecture

Figure 6.4 illustrates a unified architecture that consists of a measurement module and a prediction module. The measurement module probes the performance classes of a small number of paths and puts them in the corresponding entries of a performance matrix X. The prediction module estimates the missing entries by applying a matrix factorization technique such as DMFSGD. The estimated performance $\hat{x}_{ij}$ in $\hat{X}$ is real-valued and the predicted class can be determined by e.g. taking the sign of $\hat{x}_{ij}$.

The decentralized processing is done by using the same architectures, with no explicit construction of matrices and with each node selectively collaborating with a few other nodes, as illustrated in Figure 4.4 for RTT and Figure 4.5 for ABW in Chapter 4.

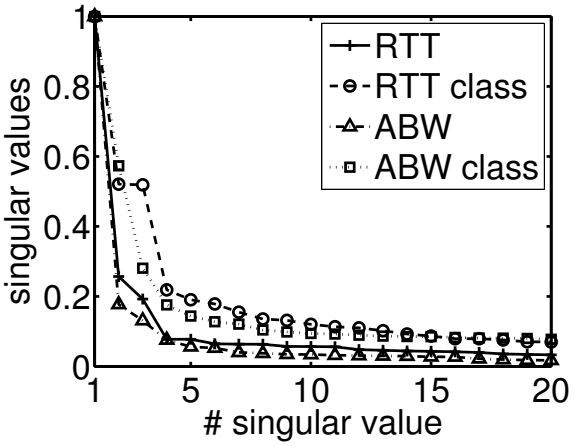


Figure 6.3: The singular values of a RTT and a ABW matrix and of their binary class matrices. The RTT and ABW matrices are extracted from the Meridian and HP-S3 dataset respectively, as described in Figure 4.2. The binary class matrices are obtained by thresholding their corresponding measurement matrices with τ equal to the median value of each dataset. The singular values are normalized so that the largest singular values of all matrices are equal to 1.

6.3.3 DMFSGD with Classification Loss Functions

The same DMFSGD algorithms in Algorithm 1 and 2 are applicable to the prediction of performance classes for RTT and for ABW, with one modification required that the loss function in the update rules of eqs. 4.4 and 4.5 and eqs. 4.6 and 4.7 taking either the hinge or the logistic loss function.

The gradients of the hinge and the logistic loss functions are given below. Without causing any confusion, the subscripts are dropped.

- For the hinge loss function, the gradients[2] are zeros for correctly classified samples, i.e., those of $1 - xuv^T \leqslant 0$, and otherwise

$$\frac{\partial l(x, uv^T)}{\partial u} = -xv, \tag{6.3}$$

$$\frac{\partial l(x, uv^T)}{\partial v} = -xu, \tag{6.4}$$

- For the logistic loss function,

$$\frac{\partial l(x, uv^T)}{\partial u} = -\frac{xv}{1 + e^{xuv^T}}, \tag{6.5}$$

$$\frac{\partial l(x, uv^T)}{\partial v} = -\frac{xu}{1 + e^{xuv^T}}, \tag{6.6}$$

[2]As the hinge loss function is not differentiable, the gradient is approximated by the subgradient [5].

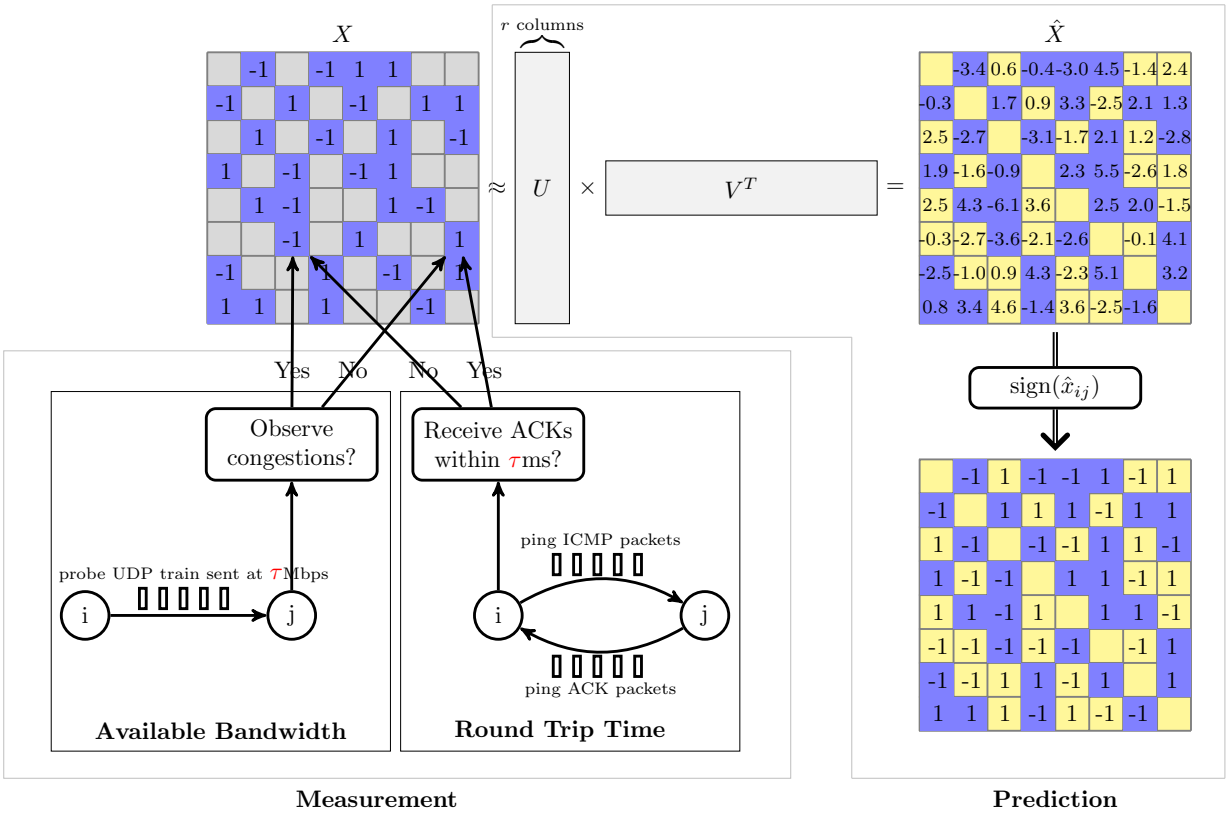


Figure 6.4: Architecture of class-based network performance measurement and prediction.

By inserting the above gradient functions in eqs. 4.4 and 4.5 and eqs. 4.6 and 4.7, the DMFSGD algorithms in Algorithm 1 and 2 can then be applied.

Unlike network distance prediction in Chapter 5 where distance measurements are noisy and take values in a large range which differs from dataset to dataset, the inputs for binary classification take only one of the two possible values of 1 and -1. This difference makes the DMFSGD algorithms here much less sensitive to the parameters. For example, a constant learning rate is found to work well for different datasets, shown in Section 6.4.2, which renders the minibatch and line-search extensions in Algorithm 4 for network distance prediction unnecessary.

6.4 Experiments and Evaluations

This section evaluates the accuracy of DMFSGD for binary classification. In addition, the robustness against erroneous measurement and the applicability on peer selection are also studied.

6.4.1 Datasets and Evaluation Criteria

The evaluations were performed on two RTT datasets of Harvard and Meridian, which are described in Section 5.5.1 in Chapter 5, and one ABW dataset, called HP-S3[3], which contains ABW measurements between 459 network nodes collected using the pathchirp tool [89]. As the raw dataset is highly sparse with 55% missing data, 231 nodes were extracted to construct a dense ABW matrix with 4% missing entries.

Different from the experiments in Section 5.5 in Chapter 5, the passivity in the measurement of the Harvard dataset is ignored in this chapter. That is, the dynamic RTT measurements are assumed to be acquired "actively", and DMFSGD is thus run on the Harvard dataset with the same neighbor selection procedure whereby each node selectively communicates with k neighbors. This allows us to study how DMFSGD is sensitive to the random selection of the paths to be monitored.

The following classification evaluation criteria [6] were used.

ROC A Receiver Operating Characteristic (ROC) curve is a graphical plot of the true positive rates (TPR) versus the false positive rates (FPR) for a binary classifier as its discrimination threshold is varied.

AUC The AUC is the area under the ROC curve. As the TPR and the FPR range from 0 to 1, the maximal possible value of AUC is 1 which means a perfect classification. In practice, AUC is smaller than 1. The closer it is to 1, the better.

Precision-Recall The precision for a class is the number of true positives divided by the total number of elements labeled as belonging to the positive class, i.e. the sum of true positives and false positives, and the recall for a class is equal to the TPR.

More precisely, the ROC and Precision-Recall curves are obtained by varying a discrimination threshold τ_c when deciding the classes from $\hat{x}_{ij}$'s. For a given τ_c, $\hat{x}_{ij}$ is turned into 1 if $\hat{x}_{ij} > \tau_c$

[3] I would like to thank Dr. Venugopalan Ramasubramanian for sharing this dataset.

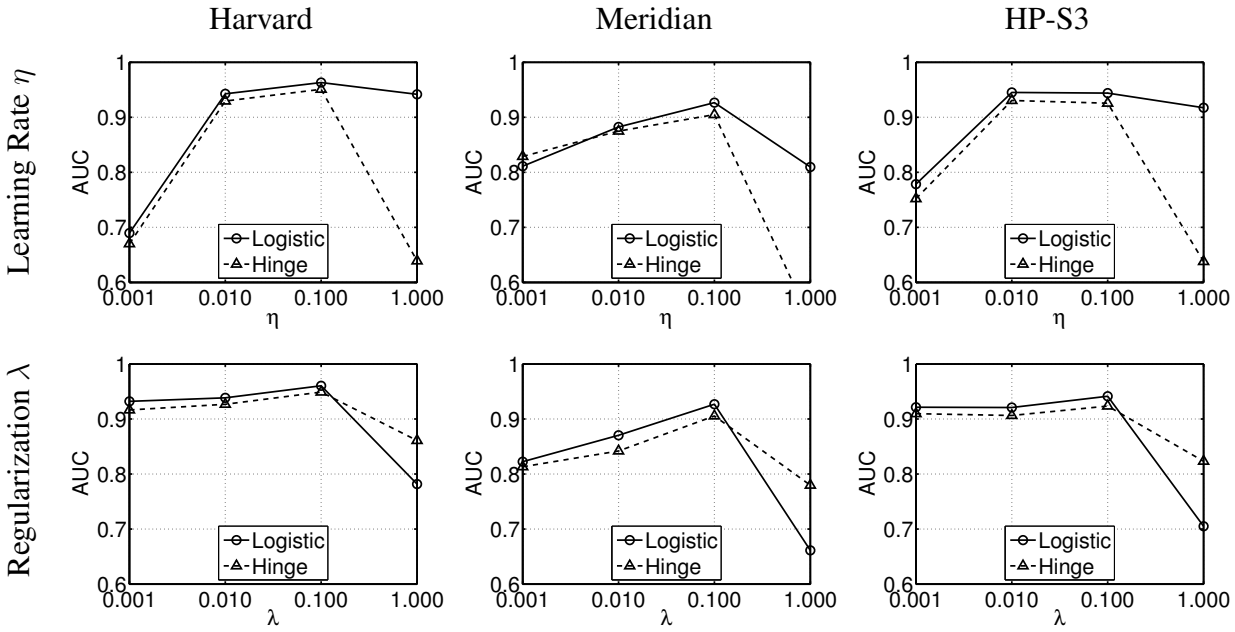


Figure 6.5: AUCs under different η's and λ's on different datasets. The first row shows the impact of η under $\lambda = 0.1$ and the second row shows the impact of λ under $\eta = 0.1$. $r = 10$ in this figure. $k = 10$, 32 and 10 for the Harvard, Meridian and HP-S3 datasets respectively. τ is set to the median value of each dataset, i.e. $\tau = 132$ms for Harvard, 56ms for Meridian and 43Mbps for HP-S3.

and into -1 otherwise. Then, the true positive rate, false positive rate and precision for the given τ_c can be computed by comparing x_{ij}'s and the binarized $\hat{x}_{ij}$'s. The ROC and Precision-Recall curves are then obtained by varying τ_c from $-\infty$ to $+\infty$. These evaluation criteria are interesting and commonly used because they show the prediction accuracies under different τ_c's. Note that τ_c is only used for plotting the ROC and Precision-Recall curves.

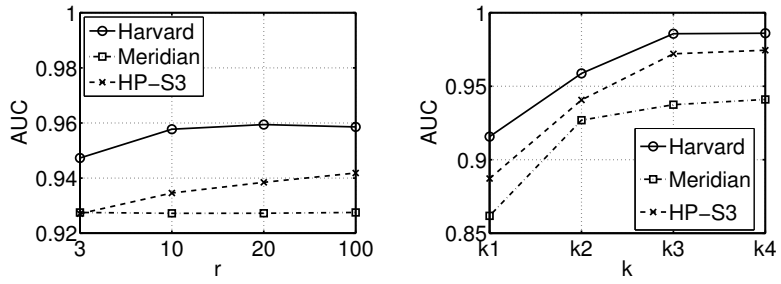
6.4.2 Impact of Parameters

DMFSGD for binary classification has four common parameters, including **learning rate** η, **regularization coefficient** λ, **rank** r and **neighbor number** k, and two additional parameters of **loss function** l (hinge or logistic) and **classification threshold** τ. The impact of the four common parameters are discussed in Section 5.5.3 in Chapter 5.

Experiments and Observations

In the first experiment, different configurations of η and λ under different loss functions were tested, shown in Figure 6.5. It can be seen that $\lambda = 0.1$ and $\eta = 0.1$ work well for all three datasets and that the logistic loss function outperforms the hinge loss function in most cases. Thus, unless stated otherwise, $\lambda = 0.1$, $\eta = 0.1$ and the logistic loss function are used by default.

In the second experiment, different configurations of r and k were tested, shown in Figures 6.6. It can be seen that a pair of relatively small k and r can already provide sufficient classifi-



(a) Rank r (b) Neighbor number k

Figure 6.6: AUCs under different k's and r's on different datasets. The left plot shows the impact of r under $k = 10$ for Harvard, 32 for Meridian and 10 for HP-S3. The right plot shows the impact of k under $r = 10$ for all datasets. The experimented k's are $k_1 = 5$, $k_2 = 10$, $k_3 = 30$ and $k_4 = 50$ for both Harvard and HP-S3 and $k_1 = 16$, $k_2 = 32$, $k_3 = 64$ and $k_4 = 128$ for Meridian. τ in the left and middle plots is set to the median value of each dataset.

cation accuracy, and further increasing k and r is either costly or worthless. Thus, unless stated otherwise, $r = 10$ and $k = 10$, 32 and 10 for Harvard, Meridian and HP-S3 respectively are used by default for all datasets. These choices of k for the different datasets lead to about $1 - 5\%$ available measurements.

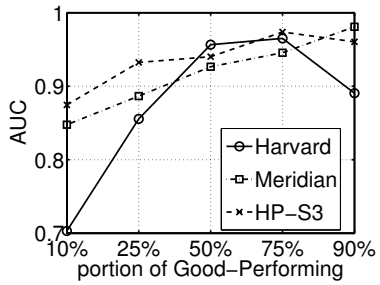
Importantly, the classification threshold τ significantly affects the proportions of the two classes, shown in Table 6.1, which in turn have some impacts on the prediction accuracy, shown in Figure 6.7. In practice τ should be determined according to the requirements of the applications. Nevertheless, τ is set to the median value of each dataset unless otherwise stated.

Table 6.1: Impact of τ on portions of "good" paths in different datasets.

"Good"%	τ		
	Harvard (ms)	Meridian (ms)	HP-S3 (Mbps)
10%	27.5	19.4	88.2
25%	59.9	36.2	72.2
50%	131.6	56.4	43.1
75%	249.6	88.1	14.4
90%	324.2	155.2	10.4

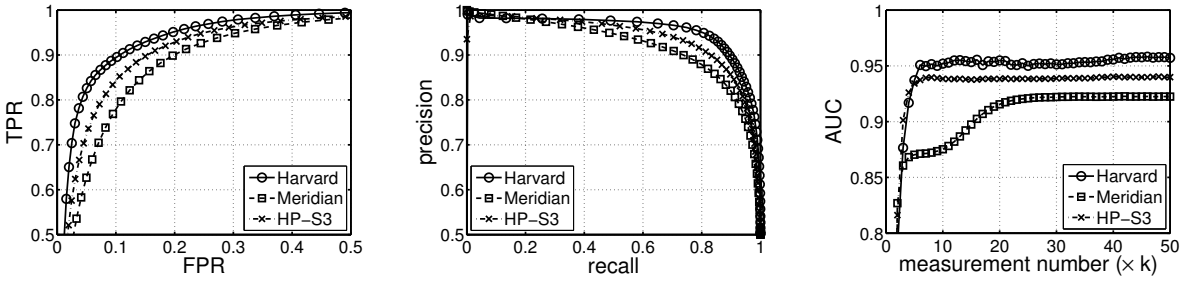
Discussions

The default parameter configuration of $\lambda = 0.1$, $\eta = 0.1$, $r = 10$ and the logistic loss function is not guaranteed to be optimal for different k's and τ's and on different datasets. However, as discussed earlier in Section 5.5.3 in Chapter 5, the fine tuning of parameters is difficult, if not



(a) Classification threshold τ

Figure 6.7: AUCs under different τ's on different datasets. The plot shows the impact of τ under $r = 10$ for all datasets and $k = 10$ for Harvard, 32 for Meridian and 10 for HP-S3. The experimented τ's for different datasets are listed in Table 6.1 to generate different portions of "good" paths.



(a) ROC (b) Precision-Recall (c) AUC

Figure 6.8: The accuracy of class-based prediction by DMFSGD on different datasets under the default parameter configuration. The rightmost plot shows the AUC improvements with respect to the average measurement number used by each node.

impossible, for network applications due to the dynamics of the measurements and the decentralized processing.

Figure 6.8 shows that the recommended default parameters produce fairly accurate results on all three datasets which are largely different from each other. The insensitivity to the parameters is probably because the inputs are binary classes and take values of either 1 or -1 regardless of the actual metric and values. The rightmost plot in Figure 6.8 illustrates the convergence speeds in terms of the AUC improvements with respect to the average measurement number per node, i.e. the total number of measurements used by all nodes divided by the number of nodes. It can be seen that the DMFSGD algorithms converge fast after each node probes, on average, no more than $20 \times k$ measurements from its k neighbors. Table 6.2 shows the accuracy rates, i.e., the percentage of the correct predictions, and the confusion matrices, computed by taking the sign of $\hat{x}_{ij}$'s and then comparing with the corresponding x_{ij}'s. In these confusion matrices, each column represents the predicted classes, while each row represents the actual classes. Thus, the off-diagonal entries represent "confusions" or mis-classifications. These measures show the accuracy of DMFSGD under the default parameters.

Table 6.2: Confusion Matrices of Binary Classification for Different Datasets

			Predicted	
Harvard	Accuracy=89.4%		"Good"	"Bad"
	Actual	"Good"	93.6%	6.4%
		"Bad"	14.7%	85.3%
	Accuracy=85.4%		Predicted	
Meridian			"Good"	"Bad"
	Actual	"Good"	88.5%	11.5%
		"Bad"	17.8%	82.2%
	Accuracy=87.3%		Predicted	
HP-S3			"Good"	"Bad"
	Actual	"Good"	93.5%	6.5%
		"Bad"	18.9%	81.1%

6.4.3 Robustness Against Erroneous Labels

This section evaluates the robustness of the DMFSGD algorithms against erroneous class labels which arise from inaccurate measurements due to

- inaccurate measurement techniques which particularly affect those paths with metric quantities close to τ;
- network anomaly such as attacks from malicious nodes and sudden traffic bursts which affect every path equally.

In particular, four different types of errors were simulated, including

Type 1: flip near τ. Flip randomly, with probability 0.5, the class labels of the paths with quantities within $[\tau - \delta, \tau + \delta]$.

Type 2: underestimation bias. For ABW[4], label erroneously the paths with quantities within $[\tau, \tau + \delta]$ as "bad".

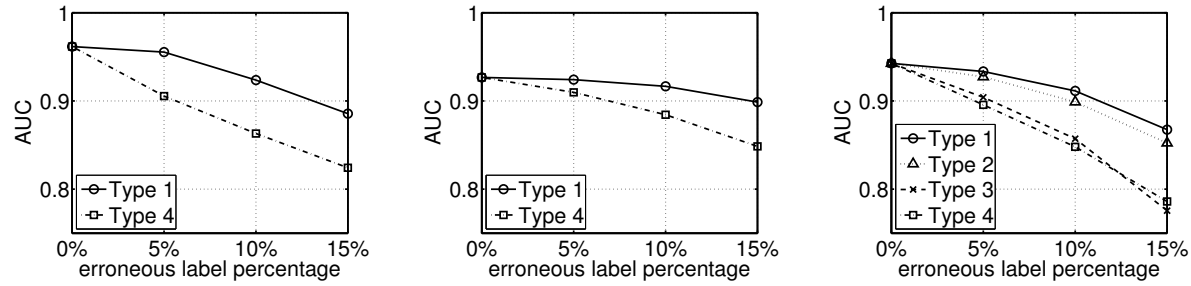
Type 3: flip randomly. For ABW[5], choose randomly $p\%$ paths and flip their labels.

Type 4: Good-to-Bad. Choose randomly $p\%$ "good" paths and label them as "bad".

In the experiments, all four types of errors were simulated for the HP-S3 dataset and the errors of Type 1 and 4 were simulated for the Harvard and Meridian datasets, as shown in Figure 6.9. Different error levels of 5%, 10% and 15% erroneous labels were tested by setting different δ's and p's for each type of errors and for each dataset. The values of δ are shown in Table 6.3 and $p = 5$, 10 and 15.

[4]Most ABW measurement tools such as pathload and pathchirp have a tendency of underestimating ABW [62], whereas such tendency is not reported by RTT measurement tools such as ping.

[5]For most network measurements, "bad" paths are unlikely to be erroneously estimated as "good". However, ABW is probed by the sender but inferred by the target node. Thus, malicious target nodes can purposely respond with flipped class labels.



(a) Harvard (b) Meridian (c) HP-S3

Figure 6.9: Robustness of class-based prediction against erroneous class labels.

It can be seen that random errors of "flip randomly" and "Good-to-Bad" have a much larger impact on classification accuracy than errors of "flip near τ" and "underestimation bias" that only perturb paths with quantities near τ. The random errors are mostly due to network anomalies. They are therefore rarer and can be addressed by incorporating heuristics such as inferring the class labels using some consensus based on recorded historical measurements. The errors of "flip near τ" and "underestimation bias" are mostly due to the inaccuracies of the measurement tools and can generally be reduced at the cost of more probe traffics, which is less necessary due to their limited impact.

Table 6.3: The values of δ that lead to certain error levels in Figure 6.9.

error%	δ			
	Harvard (ms)	Meridian (ms)	HP-S3 (Mbps)	
	Type 1	Type 1	Type 1	Type 2
5%	24.4	5.2	3.2	2.9
10%	41.5	10.2	6.7	5.7
15%	54.7	14.8	13.2	10.0

6.4.4 Peer Selection: Optimality VS. Satisfaction

For many Internet applications such as peer-to-peer downloading and streaming, the knowledge of end-to-end network performance enables intelligent peer selection that finds for each node a satisfactory node to interact with from a number of candidates. Motivated by this demand, this section demonstrates how peer selection can benefit from the prediction of network performance, in the form of both metric values and binary classes.

To this end, each node randomly selects a set of peers from all connected nodes. Denote $\mathcal{P}_i$ the peer set of node i. For each node, the nodes in the peer set are different from those in the neighbor set, i.e., $\mathcal{P}_i$ and $\mathcal{N}_i$ do not overlap. For value-based prediction, also called regression,

peer selection is done directly by choosing for each node the predicted best-performing node in the peer set, i.e., at node i,

$$j_p = \begin{cases} \arg\min_{j \in \mathcal{P}_i} \hat{x}_{ij} & \text{for RTT} \\ \arg\max_{j \in \mathcal{P}_i} \hat{x}_{ij} & \text{for ABW} \end{cases}. \tag{6.7}$$

For class-based prediction, also called classification, peer selection is done by selecting the peer in the peer set which is the most likely to be "good". In classification, the absolute value of the output $\hat{x}_{ij} = u_i v_j^T$ defines the margin that reflects how confident the estimate is. Thus, the peer that is the most likely to be "good" is the one with the largest predicted value, i.e., at node i,

$$j_p = \arg\max_{j \in \mathcal{P}_i} \hat{x}_{ij}. \tag{6.8}$$

Note that the output $\hat{x}_{ij}$ by classification is directly used without taking its sign or thresholding it by some τ_c. To demonstrate the impact of erroneous labels to peer selection, two types of errors were simulated including 10% "flip near τ" and 5% "Good-to-Bad", leading to overall 15% erroneous labels for all datasets. The random peer selection is used as a baseline method for comparison.

The commonly-used evaluation criterion for peer selection is the stretch [90], defined as

$$s_i = \frac{x_{i\bullet}}{x_{i\circ}}, \tag{6.9}$$

where $\bullet$ is the id of the selected peer, $\circ$ is that of the true best-performing peer in the peer set of node i and $x_{i\bullet}$ and $x_{i\circ}$ are the measured quantities of some performance metric. s_i is larger than 1 for RTT and smaller than 1 for ABW. The closer s_i is to 1, the better.

The stretch reflects the optimality of peer selection, shown in the first row of Figure 6.10. As expected, peer selection based on both classification and regression outperforms random peer selection and the best optimality is achieved by using regression. Indeed, classification seeks to provide satisfactory instead of optimal services. To demonstrate this, the average percentage of unsatisfied nodes is calculated and plotted, defined as the nodes that select wrongly "bad" peers when there are "good" peers available in the peer sets, shown in the second row of Figure 6.10. It can be seen that in terms of satisfaction, classification is sufficient to provide satisfactory services with about on average 10% unsatisfied nodes and as large as 15% erroneous labels only degrade the performance of peer selection by less than 5% for all datasets.

Note that regression achieved on HP-S3 a marginally better performance than classification, less than 5%, on the percentage of unsatisfied nodes, however at the cost of much more measurement overheads in order to get precise ABW values. It should also be noted that always selecting best-connected nodes doesn't make efficient use of the overall capacity of the networks and may cause congestions and overloading to those nodes due to their popularity especially in the beginning of the services [21].

6.5 Conclusions and Discussions

This chapter presents how DMFSGD is adapted and applied for predicting end-to-end network performance classes. The success roots largely in the class-based representation of network per-

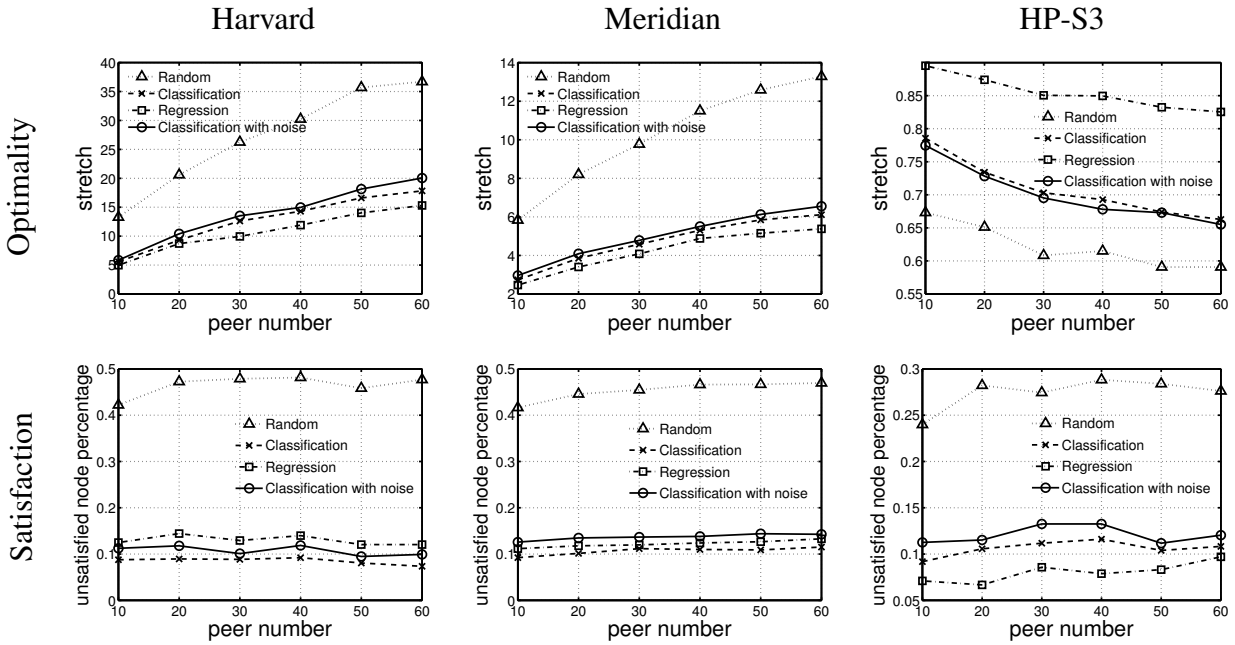


Figure 6.10: Peer selection with various numbers of peers in the peer set of each node. The top row shows the optimality of the peer selection in terms of the average stretch, and the bottom row shows the satisfaction in terms of the average percentage of unsatisfied nodes, defined as the nodes that select wrongly "bad" peers when there exist "good" peers in the peer sets. The nodes with a peer set of all "bad" peers are excluded from the calculation as no satisfactory peers can be selected.

formance, which not only lowers the measurement costs but also enables a unified treatment of various metrics. The extensive experiments demonstrate the advantages of predicting performance classes by DMFSGD, which show not only the accuracy but also the robustness against erroneous measurements. The case study on peer selection highlights the benefits of the prediction system and the usability by real Internet applications.

In practice, a binary measure is sensitive to the choice of the classification threshold τ and may be too coarse for applications. Thus, the next chapter studies a multiclass approach that classifies network performance into multiple classes, represented by ratings.

Chapter 7

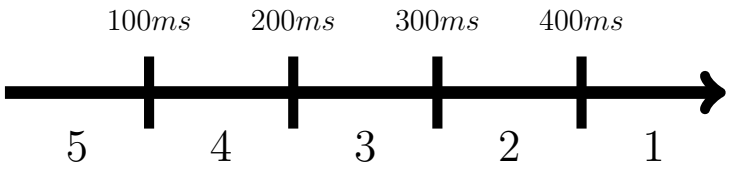
Predicting End-to-End Network Performance Ratings

This chapter introduces the ordinal rating of end-to-end network performance and the prediction of performance ratings. While the ordinal rating has similar properties as the binary classification including low measurement cost and unifying various metrics, it is more fine-grained and provides richer information. The prediction of ordinal ratings can be done by the same matrix completion framework as described in Chapter 4. By adopting the popular 5-star rating system, the rating prediction problem can potentially be solved by the solution that won the Netflix prize. As this solution integrated various matrix factorization models, a particular interest of this chapter is to investigate the applicability of these matrix factorization models to the rating prediction problem.

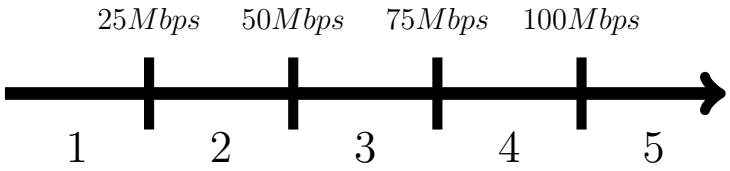
7.1 Ordinal Rating of Network Performance

Rating network performance amounts to assigning to a path an ordinal number in the range of $\{1, R\}$, with a larger value indicating better performance, according to some metric. Commonly, the 5-star rating system with $R = 5$ has been widely used in many Internet sites and services including Amazon, Netflix and iTunes. This 5-star rating system is also adopted in this chapter. The different levels of rating indicate qualitatively how well network paths would perform, i.e., 1–"very poor", 2–"poor", 3–"ordinary", 4–"good" and 5–"very good".

Similar as binary classes, ratings can be acquired by thresholding that partitions the range of the metric into R bins using $R - 1$ thresholds, denoted by $\tau = \{\tau_1, \ldots, \tau_{R-1}\}$, and determines to which bins metric values belong, as illustrated in Figure 7.1. Note that when $R = 2$, ordinal rating degenerates to binary classification. The thresholds can be chosen evenly or unevenly according to the requirements of the applications. Clearly, rating a network path is much cheaper than measuring the exact metric value, as it only requires to determine if the metric value is within a certain range defined by the thresholds. The cost reduction is particularly significant for the metric of available bandwidth (ABW) due to its measurement methodology, as described in Section 6.2 in Chapter 6.



(a) RTT



(b) ABW

Figure 7.1: The quantization of metric values into ratings on a scale of $\{1, 5\}$. The thresholds are chosen as example.

Thus, ordinal ratings are performance measures that are in between metric values and binary classes.

- On the one hand, ratings are, like metric values, more fine-grained and thus more informative than binary classes.
- On the other hand, ratings are, like binary classes, qualitative and thus bear the same advantages over metric values, including the low measurement cost, the unification of different metrics and being stable and reflective of user experience, etc.

7.2 Predicting Ratings by Matrix Completion

7.2.1 Formulation as Matrix Completion

The rating prediction problem has the same matrix completion formulation as the one in network distance prediction in Chapter 5 and in binary classification in Chapter 6. Recall that the objective function is given in eq. 4.1 in Chapter 4, which is

$$\{(u_i, v_i), i = 1, \ldots, n\} = \arg\min \sum_{(i,j)\in\Omega} l(x_{ij}, u_i v_j^T) + \lambda \sum_{i=1}^{n} u_i u_i^T + \lambda \sum_{i=1}^{n} v_i v_i^T.$$

One main difference is that the performance measure x_{ij} here takes ordinal numbers on a scale of $\{1, 5\}$. As mentioned earlier in Section 4.1.1 in Chapter 4, the matrix to be completed is required to have a low-rank characteristic. To show that the low-rank assumption holds for matrices of performance ratings, Figure 7.2 plots the singular values of a RTT and a ABW matrix and of the related rating matrices. It can be seen that the singular values of all matrices decrease fast, indicating a strong low-rank characteristic.

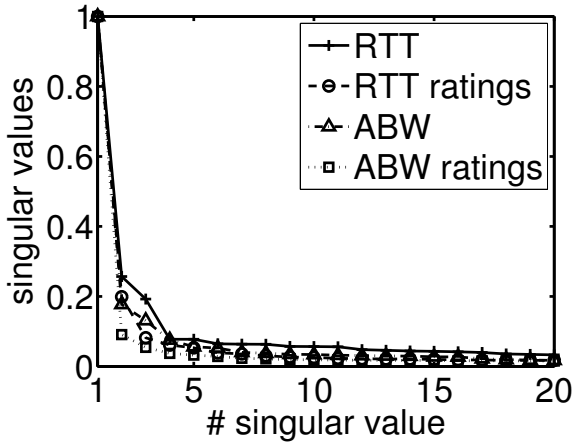


Figure 7.2: The singular values of a RTT and a ABW matrix and of their rating matrices. The RTT and ABW matrices are extracted from the Meridian and HP-S3 dataset respectively, as described in Figure 4.2. The rating matrices are obtained by thresholding their corresponding measurement matrices with $\tau = \{20\%, 40\%, 60\%, 80\%\}$ percentiles of each dataset. The singular values are normalized so that the largest singular values of all matrices are equal to 1.

7.2.2 Applicability of the Solutions to the Netflix Prize

A particular advantage of using the popular 5-star rating system is that it has been widely used in numerous recommender systems including the one at Netflix. In particular, the Netflix prize required to predict the preference of users to movies, quantized from 1 to 5 stars [57, 4]. Great efforts have been put on solving the problem in the Netflix prize as optimally as possible, leading to various approaches and algorithms. The winner of the prize is the BellKor's Pragmatic Chaos team [57]. In the sequel, the prize-winning solution is called BPC. Thus, due to the resemblance of the two problems, a particular interest of this chapter is to investigate the applicability of the solution of BPC to the rating prediction problem.

In particular, BPC integrated two classes of techniques based on neighborhood models and on matrix factorization. Neighborhood models exploit the similarities between users and between items, calculated as correlation coefficients. For example, two users are considered to share common interests if they rate a set of items similarly. Meanwhile, two items are considered similar if they are given similar ratings by a set of users. Interesting as they are, calculating similarities requires a sufficient number of ratings which may not be available. In the context of network performance prediction, to compute the similarities between two network nodes, the two nodes must probe a number of common nodes, i.e., the neighbor sets of the two nodes have to overlap with a sufficient degree. This is almost impossible, as in the decentralized system architecture, the neighbors of a node is often randomly selected with no control at all.

Thus, the focus below is the applicability of the matrix factorization models in BPC.

7.3 Various Matrix Factorization Models

This section discusses various matrix factorization (MF) models that were integrated in BPC [57], including regularized matrix factorization (RMF), max-margin matrix factorization (MMMF), non-negative matrix factorization (NMF) and the ensemble of matrix factorization (MF ensemble). These MF models were introduced in the reports of BPC [57] and in [88].

7.3.1 RMF

Regularized matrix factorization (RMF) is essentially the standard matrix factorization model described in Section 3.2 in Chapter 3, with the L_2 loss function adopted in eq. 3.11, leading to the following the objective function

$$\min \sum_{(i,j)\in\Omega} (x_{ij} - u_i v_j^T)^2 + \lambda \sum_{i=1}^{n} (u_i u_i^T + v_i v_i^T). \tag{7.1}$$

Basically, RMF ignores the fact that the performance measure x_{ij} takes an ordinal number. The predicted performance $\hat{x}_{ij} = u_i v_j^T$ is real-valued and has to be rounded to the closest integer in the range of $\{1, R\}$.

7.3.2 MMMF

Max-margin matrix factorization (MMMF) [67] takes into account that the inputs are ordinal numbers and solves the rating prediction problem by ordinal regression.

As RMF, the predicted performance $\hat{x}_{ij}$ is calculated by $u_i v_j^T$. However, the real-valued estimate $\hat{x}_{ij}$ is then related to the ordinal rating x_{ij} by using $R-1$ thresholds, denoted by $\theta = \{\theta_1, \ldots, \theta_{R-1}\}$. Thus, MMMF requires the following constraint to be satisfied for each x_{ij}, $(i,j) \in \Omega$,

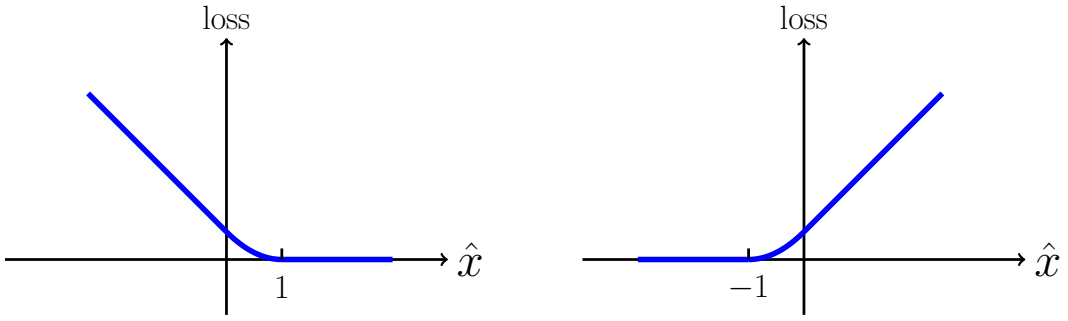
$$\theta_{c-1} < \hat{x}_{ij} = u_i v_j^T < \theta_c, \text{ for } x_{ij} = c, \ 1 \leqslant c \leqslant R. \tag{7.2}$$

For simplicity of notation $\theta_0 = -\infty$ and $\theta_R = +\infty$. In words, the value of $\hat{x}_{ij}$ does not matter, as long as it falls in the range of $\{\theta_{c-1}, \theta_c\}$ for $x_{ij} = c$, $1 \leqslant c \leqslant R$. The threshold θ is a parameter that can in theory be learned from the data. However, to simplify the problem, $\theta = \{\theta_1, \ldots, \theta_{R-1}\}$ is fixed and set to $\{1.5, 2.5, 3.5, 4.5\}$ for $R = 5$. Thus, the constraint in eq. 7.2 means that, for example, if $x_{ij} = 2$, then it is required that $1.5 < \hat{x}_{ij} < 2.5$ so that $\hat{x}_{ij}$ will be rounded to 2. Whether $\hat{x}_{ij}$ is 2, 2.2 or 1.6 makes no difference.

In practice, it is impossible to have eq. 7.2 satisfied for every x_{ij}. Thus, the violations of the constraints are penalized and the following objective function is to be minimized, given by

$$\min \sum_{(i,j)\in\Omega} \sum_{c=1}^{R-1} l(T_{ij}^c, \theta_c - u_i v_j^T) + \lambda \sum_{i=1}^{n} (u_i u_i^T + v_i v_i^T), \tag{7.3}$$

where $T_{ij}^c = 1$ if $x_{ij} \geqslant c$ and -1 otherwise. Essentially, the objective function in eq. 7.3 consists of a number of binary classification losses, each of which compares an estimate $\hat{x}_{ij}$ with a threshold θ_c in $\{\theta_1, \ldots, \theta_{R-1}\}$.



(a) $x = 1$ (b) $x = -1$

Figure 7.3: The smooth hinge loss function. In this loss function, x takes a discrete value of either 1 or -1, as in the hinge and logistic loss function in Figure 6.2. Note that the smooth hinge loss function is smooth and thus differentiable.

For example, for $R = 5$, $\theta = \{1.5, 2.5, 3.5, 4.5\}$ and x_{ij}=2, it is required that $\hat{x}_{ij} > 1.5$ and $\hat{x}_{ij} < 2.5$, $\hat{x}_{ij} < 3.5$, $\hat{x}_{ij} < 4.5$. Each violation of these four constraints is penalized by $l(T_{ij}^c, \theta_c - u_i v_j^T)$, with

$$T_{ij}^c = \begin{cases} -1 & \text{for } c = 1 \\ 1 & \text{for } 2 \leqslant c \leqslant 4 \end{cases} \tag{7.4}$$

indicating the correct sign of $(\theta_c - u_i v_j^T)$.

The loss function l can be any classification loss function such as the hinge and the logistic loss function described in Section 6.3.1 in Chapter 6. In [67], a modified hinge loss function, so-called smooth hinge loss function, was proposed, given by

$$l(x, \hat{x}) = \begin{cases} 0 & \text{if } x\hat{x} \geqslant 1 \\ \frac{1}{2}(1 - x\hat{x})^2 & \text{if } 0 < x\hat{x} < 1 \\ \frac{1}{2} - x\hat{x} & \text{if } x\hat{x} \leqslant 0 \end{cases}. \tag{7.5}$$

Figure 7.3 shows the smooth hinge loss function. Comparing with the hinge loss function in eq. 6.1, the smooth hinge loss function is smooth and thus differentiable. Following the work in [67] which was also used in BPC, the smooth hinge loss function is adopted here for ordinal rating.

7.3.3 NMF

Non-negative matrix factorization (NMF) [41] incorporates an additional constraint that all entries in (U, V) have to be non-negative so as to ensure the non-negativity of $\hat{X} = UV^T$. NMF is described in Section 5.3.4 in Chapter 5. However, when dealing with ordinal ratings, NMF often uses the divergence [6] to measure the difference between X and $\hat{X}$, defined as

$$D(X||\hat{X}) = \sum_{ij \in \Omega} (x_{ij} \log \frac{x_{ij}}{\hat{x}_{ij}} - x_{ij} + \hat{x}_{ij}). \tag{7.6}$$

Note that although the divergence reflects the dissimilarity between two variables, it is not a distance or a metric because it is asymmetric, i.e., $D(X||Y) \neq D(Y||X)$.

Thus, NMF minimizes an objective function of the following form

$$\min D(X||UV^T) + \lambda \sum_{i=1}^{n} (u_i u_i^T + v_i v_i^T). \quad (7.7)$$
$$\text{s.t. } U \geqslant 0,\ V \geqslant 0$$

As RMF, $\hat{x}_{ij} = u_i v_j^T$ is also real-valued and has to be rounded to the closest integer in the range of $\{1, R\}$.

7.3.4 MF ENSEMBLE

The success of BPC built largely on the idea of the ensemble method which learns multiple models and combines their outputs for prediction. The reason why ensemble methods can work is that they can reduce the variance of learning algorithms [24, 27]. In machine learning, usually several different models can give similar accuracy on the training data but perform unevenly on the unseen data. In this case, a simple vote or average of the outputs of these models can reduce the variance of the predictions. More intuitively, the power of ensemble methods comes from the "wisdom of the crowd", which says that a large group's aggregated answer to a question is generally found to be as good as, and often better than, the answer given by any of the individuals within the group [77].

Thus, the above RMF, MMMF and NMF are combined in a MF ensemble approach. The final prediction result is the average of the predictions by different MF models, as described in the reports of BPC [57] and in [88].

7.3.5 Inference By Stochastic Gradient Descent

The above MF models are all solved by stochastic gradient descent (SGD), with the basic DMFSGD algorithm in Algorithm 1 for RTT and in Algorithm 2 for ABW directly usable. The gradients of the loss function in the update rules in eqs. 4.4 and 4.5 for RTT and in eqs. 4.6 and 4.7 for ABW can be easily derived according to the choice of the loss function for each MF model. Similar to binary classification, the minibatch and line-search extensions in the DMFSGD algorithm in Algorithm 4 are not necessary, as the inputs here take only one of a few possible values, for example, in the range of $\{1, 5\}$.

7.4 Experiments and Evaluations

The evaluations were performed on the two RTT datasets of Harvard and Meridian and the ABW dataset of HP-S3, as described in Section 6.4.1 in Chapter 6. The common evaluation criterion, Rooted Mean Square Error (RMSE), used for recommender systems and for the Netflix prize is

adopted, defined as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{n}}. \tag{7.8}$$

Note that the smaller RMSE is, the better.

7.4.1 Obtaining Ratings

The first step is to obtain ratings on the scale of $\{1, 5\}$ from the metric values in the three datasets. To this end, the range of the metric is partitioned by the rating threshold $\tau = \{\tau_1, \ldots, \tau_4\}$. τ is set by two strategies:

- Strategy 1: set τ by the $\{20\%, 40\%, 60\%, 80\%\}$ percentiles of each dataset.
  - Harvard: $\tau = \{48.8, 92.2, 177.2, 280.3\}$ms
  - Meridian: $\tau = \{31.6, 47.3, 68.6, 97.9\}$ms
  - HP-S3: $\tau = \{12.7, 34.5, 48.8, 77.9\}$Mbps
- Strategy 2: partition evenly the range between 0 and a large value selected for each dataset.
  - Harvard: $\tau = \{75, 150, 225, 300\}$ms
  - Meridian: $\tau = \{25, 50, 75, 100\}$ms
  - HP-S3: $\tau = \{20, 40, 60, 80\}$Mbps

Note that Strategy 2 produces quite unbalanced portions of ratings on each dataset.

7.4.2 Comparison of Different MF Models

In the simulations, the random neighbor selection procedure, described in Section 5.3.2 in Chapter 5 and used in all the experiments in Section 6.4 in Chapter 6, was adopted whereby each node randomly selects k neighbors to probe. As in Section 6.4 in Chapter 6, $k = 10$ for Harvard and HP-S3 and $k = 32$ for Meridian lead to about $1 - 5\%$ available measurements. Another important parameter in MF is the rank r. As in Section 6.4 in Chapter 6, $r = 10$ for all datasets.

The three MF models including RMF, MMMF and NMF were all solved by SGD. The learning rate of SGD η equals 0.05 and the regularization coefficient λ equals 0.1. Note that the parameters are not fine-tuned for each dataset and for each model, as it is impossible for the decentralized processing on a network. Similar to binary classification, MF for ordinal rating is not very sensitive to the parameters as the inputs are ordinal numbers on the scale of $\{1, 5\}$, regardless of the actual metric and values. For MF ensemble, following the procedure in [88], several predictors were generated for each MF model using different parameters, i.e., the rank r ranges from 10 to 100 and the regularization λ changes from 0.01 to 1. Although maintaining multiple predictors in parallel may not be practical, MF ensemble produces the (nearly) optimal accuracy that could be achieved based on MF in a centralized manner.

Table 7.1 and 7.2 show the RMSE achieved using different MF models and different τ-setting strategies. Particularly, the following observations were made. First, RMF generally performs better than MMMF and NMF and MF ensemble performs the best. Second, the improvement of

MF ensemble over RMF is only marginal, which is not considered worth the extra cost. Third, by comparing Table 7.1 and 7.2, it is clear that different settings of τ have some impacts to the accuracy, which need to be further studied. Nevertheless, Strategy 1 is adopted in the sequel.

Table 7.1: RMSE with τ set by strategy 1

τ	Harvard	Meridian	HP-S3
RMF	0.9340	0.8306	0.6754
MMMF	0.9688	0.8634	0.6862
NMF	0.9772	0.9042	0.6820
MF Ensemble	0.9205	0.8214	0.6611

Table 7.2: RMSE with τ set by strategy 2

τ	Harvard	Meridian	HP-S3
RMF	0.9198	0.7761	0.6669
MMMF	0.9193	0.8099	0.6697
NMF	0.9316	0.8286	0.6742
MF Ensemble	0.9043	0.7658	0.6527

It is worth mentioning that for the dataset in the Netflix prize, the RMSE achieved by the Netflix's algorithm Cinematch is 0.9525 and that by BPC is 0.8567. While the RMSEs on different datasets are not comparable, it shows that in practice, the prediction with an accuracy of the RMSE less than 1 for ratings on a scale of $\{1, 5\}$ is already accurate enough to be used by applications. Thus, by trading off between the accuracy and the practicability, the RMF model is adopted by default in the rating prediction system. Table 7.3 shows the confusion matrices achieved by RMF on the three datasets. In these matrices, each column represents the predicted ratings, while each row represents the actual ratings. Thus, the off-diagonal entries represent "confusions" or mis-ratings. It can be seen that while there are mis-ratings, few have an error of $|x_{ij} - \hat{x}_{ij}| > 1$, which means that the mis-ratings are under control.

7.4.3 Peer Selection: Optimality

Furthermore, the task of peer selection was performed to demonstrate the usability of network performance prediction, based on ratings of $\{1, 5\}$ in this chapter, based on binary classes of "good" and "bad" in Chapter 6, and based on metric values in Chapter 5. As the peer selection procedure in Section 6.4.4 in Chapter 6, each node randomly selects a set of peers from all available nodes in the network. Each node then chooses one peer from its peer set, and the optimality of the peer selection is calculated by the stretch, given in eq. 6.9. Recall that the stretch is larger than 1 for RTT and smaller than 1 for ABW. The closer it is to 1, the better.

Table 7.3: Confusion Matrices of Ordinal Rating for Different Datasets

Harvard

	1	2	3	4	5
1	0.68	0.28	0.02	0.01	0.00
2	0.18	0.60	0.20	0.01	0.00
3	0.03	0.13	0.66	0.17	0.00
4	0.03	0.04	0.16	0.67	0.10
5	0.04	0.03	0.05	0.43	0.46

Meridian

	1	2	3	4	5
1	0.78	0.18	0.03	0.01	0.00
2	0.08	0.59	0.29	0.03	0.00
3	0.01	0.18	0.60	0.20	0.01
4	0.01	0.03	0.33	0.59	0.04
5	0.01	0.01	0.12	0.59	0.27

HP-S3

	1	2	3	4	5
1	0.92	0.06	0.01	0.00	0.00
2	0.11	0.65	0.22	0.02	0.00
3	0.01	0.20	0.68	0.11	0.00
4	0.00	0.03	0.33	0.58	0.06
5	0.00	0.01	0.10	0.55	0.34

Figure 7.4 shows the stretch of peer selection achieved based on value-based prediction, class-based prediction and rating-based prediction. Random peer selection is used as a baseline method for comparison. It can be seen that on the optimality, value-based prediction performs the best and the performance by rating-based prediction is better than that of class-based prediction. This shows that the rating information is a good comprise between metric values and binary classes. On the one hand, ratings are more informative than binary classes and allow to find better-performing paths. On the other hand, ratings are qualitative and thus require less measurement costs than metric values.

7.5 Conclusions and Discussions

This chapter presents how to predict end-to-end network performance ratings by matrix completion. In particular, different matrix factorization models used in recommender systems, particularly in the solution that won the Netflix prize, are investigated, which shows that the simple regularized matrix factorization can already produce good accuracies with relatively low computational cost. The case study on peer selection highlights the benefits of the ordinal rating, being more fine-grained than the binary classification and meanwhile having lower measurement cost than the metric value.

The study in this chapter is incomplete. For example, how the prediction accuracy is affected

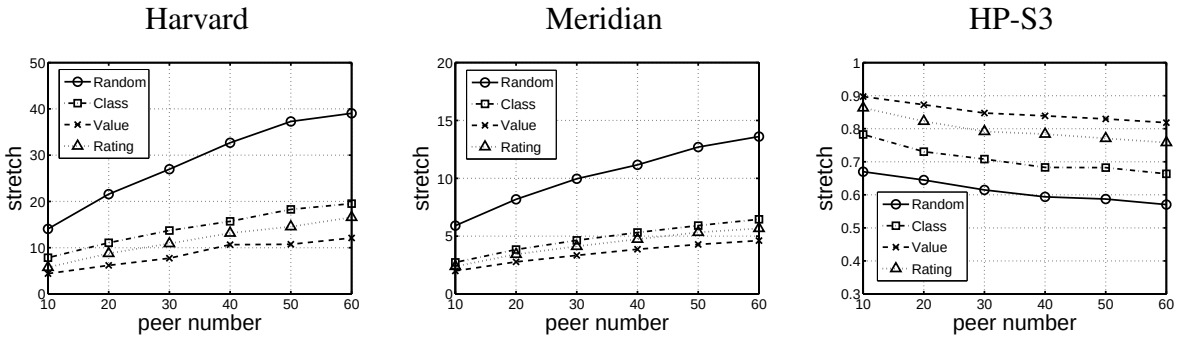


Figure 7.4: Peer selection with various numbers of peers in the peer set of each node.

by erroneous measurements is not studied, because the noise model for the rating is more complicated and needs to be carefully designed. The satisfaction of peer selection based on ratings is not studied either, because it is less clear how the satisfaction of a node to the path selection should be defined by using the rating information. These are left as future work.

Chapter 8

Conclusions and Future Work

This chapter summarizes the work and points out directions for future research.

8.1 Conclusions

This thesis presents the research work on approaches to end-to-end network performance prediction. The studies were motivated by the demands of the knowledge of network performance by emerging Internet services such as Content Delivery Networks (CDNs) and P2P Overlay Networks. The task is challenging due to the diversity of the Internet which has not yet been completely understood and well modeled.

The approach developed in this thesis extends previous work in the field in a number of ways. First, it has a general framework based on matrix completion by matrix factorization, which is solidly founded on the recent advances in machine learning and in mathematics. The framework requires neither structural information of the network nor geometric constraints. Instead, it exploits the spatial correlations across network measurements on different paths, which have long been observed in numerous research. Second, the approach has a fully decentralized architecture which naturally integrates the measurement methodologies of particular metrics and a stochastic optimization algorithm, namely stochastic gradient descent. By letting network nodes exchange messages with each other, the architecture relies on neither a central server nor landmark nodes. Third, the approaches address issues such as the high measurement cost of some metrics and the diverse nature of different metrics by adopting two qualitative representations of network performance, namely binary classification and ordinal rating.

A class of algorithms, so-called Decentralized Matrix Factorization by Stochastic Gradient Descent (DMFSGD), were proposed. The algorithms are flexible to deal with performance measures under various metrics and in the form of not only exact values but also binary classes and ordinal ratings, with little modification required. **This flexibility is a unique feature which distinguishes DMFSGD from all other approaches.** In addition, the algorithms are simple, computationally lightweight and accurate, allowing DMFSGD to deal with practical issues when deployed in real networks, such as measurement dynamics, where network measurements vary largely over time, and network churn, where nodes join and leave a network frequently. The ef-

fectiveness of DMFSGD has been justified on various publicly-available datasets of two metrics, namely RTT and ABW. Moreover, a case study on peer selection demonstrates the benefit of network performance prediction, based on metric values, on binary classes and on ordinal ratings, which highlights the usability of the prediction system by real Internet applications.

During the development of the work, there are important lessons learned and interesting observations made, some of which are given here.

- In using DMFSGD for network performance prediction, the assumption is that the performance matrix to be completed has a low-rank characteristics. While this assumption is realistic, empirically justified by the spectral plots in Section 4.1.1 in Chapter 4, Section 6.3.1 in Chapter 6 and Section 7.2.1 in Chapter 7, it is also convenient, enabling an easy resolution of the problem. Thus, the lesson learned is that when building a model for solving a real-world problem, it is important to take into account both the model accuracy, based on observable reality, and the model tractability, based on available resources. A good trade-off between the two is a critical criterion in model selection.

- Quantizing real metric values into discrete-valued classes and ratings is advantageous in that it not only provides useful information that is reflective of user experience on network performance, but also reduces measurement costs and unifies different metrics. However, the study on peer selection in Section 6.4.4 in Chapter 6 and in Section 7.4.3 in Chapter 7 shows that such quantification is rather coarse and hurts the optimality of peer selection. This makes binary classification and ordinal rating more interesting for ABW than for RTT, because the measurement of RTT is cheap and can sometimes be done passively. Thus, it is recommended that for RTT, the value should be either used directly or quantized into more ratings, for example on a scale of $\{1, 100\}$.

- It is mentioned in Section 4.1.2 in Chapter 4 that the problem of network performance prediction resembles the problem in recommender systems. This insight makes us wonder how a prediction system should be evaluated from the applications' point of view. For example, in recommender systems, it is only important to have the preference to those recommended items estimated correctly or accurately. Thus, for tasks such as intelligent path selection, it may not be worth pushing the overall accuracies for all paths to the limit, as only the accuracies of those recommended paths matter.

8.2 Future Work

There are several directions of future work.

- While the DMFSGD approaches prove to work well for the metric of RTT and ABW, a natural follow-up is to study how they work for other metrics, such as packet loss rate and hop-count distance, and other related problems such as network traffic estimation and topology recovery.

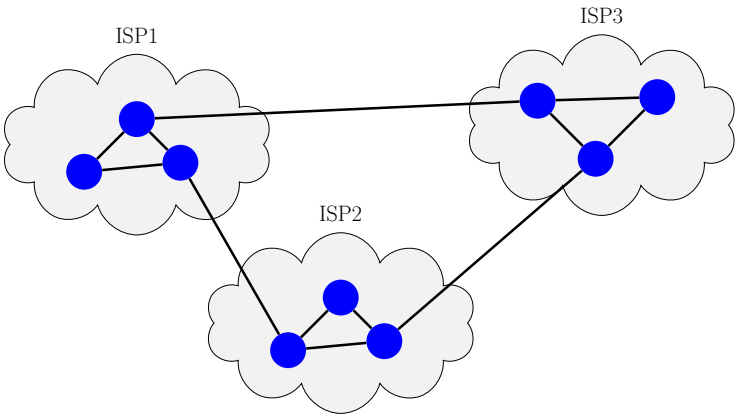


Figure 8.1: Locality-aware overlay construction.

- While it is clear that the measurement of performance classes and ratings is cheap, it would be interesting to study the cost reduction, particularly for ABW. It is expected that the amount of cost reduction depends on the classification or rating threshold. For example, for binary classification, if the metric value is close to the classification threshold τ, then it would be more costly to determine accurately whether the metric value is larger or smaller than τ.

- Finally, it would be interesting to study how the DMFSGD algorithms can help improve the performance of Internet applications. For example, in the construction of P2P overlay networks, it is desirable to enforce the locality that avoids using many connections between nodes in different Internet Service Providers (ISPs), illustrated in Figure 8.1. It has been shown that locality-aware overlay construction can reduce the cross-ISP traffic and improve the performance of services, for example increasing the download rate in P2P applications [23, 61, 19]. To this end, the knowledge of node proximity based on network performance can be exploited. It is worth studying how DMFSGD can be incorporated to enhance the locality in overlay construction and routing. In addition, I am also interested in deploying the prediction system in CDNs and in developing more effective algorithms for server selection and content replica placement.

Bibliography

[1] *matlab mdscale*. `http://www.mathworks.com/help/stats/mdscale.html`.

[2] Suman Banerjee, Timothy G. Griffin, and Marcelo Pias. The interdomain connectivity of PlanetLab nodes. In *Proc. of the Passive and Active Measurement*, Antibes Juan-les-Pins, France, April 2004.

[3] Harpal Singh Bassali, Krishnanand M. Kamath, Rajendraprasad B. Hosamani, and Lixin Gao. Hierarchy-aware algorithms for CDN proxy placement in the Internet. *Computer Communications*, 26(3):251–263, February 2003.

[4] James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *KDD Cup and Workshop at the 13th ACM SIGKDD Conference*, 2007.

[5] D.P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, New York, NJ, USA, 2006.

[7] Ingwer Borg and Patrick Groenen. *Modern multidimensional scaling : theory and applications*. Springer, 2005.

[8] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, 1998.

[9] C. J. Bovy, H. T. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal, and P. van Mieghem. Analysis of end-to-end delay measurements in Internet. In *Proc. of the Passive and Active Measurement*, Fort Collins, CO, USA, March 2002.

[10] A. M. Buchanan and A. W. Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. In *Computer Vision and Pattern Recognition*, 2005.

[11] Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali. *Content Delivery Networks*. Springer, 2008.

[12] Jian-Feng Cai, Emmanuel J. Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[13] E. J. Candès and Y. Plan. Matrix completion with noise. *Proc. of the IEEE*, 98(6), 2010.

[14] E. J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

[15] Jacob Chakareski and Pascal Frossard. Delay-based overlay construction in p2p video broadcast. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1985–1988, 2009.

[16] Yan Chen, David Bindel, Hanhee Song, and Randy H. Katz. An algebraic approach to practical and scalable overlay network monitoring. *SIGCOMM Comput. Commun. Rev.*, 34(4):55–66, August 2004.

[17] Yan Chen, Randy H. Katz, and John Kubiatowicz. Dynamic replica placement for scalable content delivery. In *the First International Workshop on Peer-to-Peer Systems*, pages 306–318, 2002.

[18] Yang Chen, Xiao Wang, Xiaoxiao Song, Eng Keong Lua, Cong Shi, Xiaohan Zhao, Beixing Deng, and Xing Li. Phoenix: Towards an accurate, practical and decentralized network coordinate system. In *Proc. of IFIP Networking Conference*, Aachen, Germany, May 2009.

[19] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 363–374, 2008.

[20] D. B. Chua, E. D. Kolaczyk, and M. Crovella. Network kriging. *IEEE Journal on Selected Areas in Communications*, 24(12):2263–2272, December 2006.

[21] Mark Crovella and Balachander Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. John Wiley & Sons, Inc., New York, NY, USA, 2006.

[22] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. of ACM SIGCOMM*, Portland, OR, USA, August 2004.

[23] Peter J. Denning. The locality principle. *Commun. ACM*, 48(7):19–24, July 2005.

[24] Thomas G. Dietterich. Ensemble learning. In *The Handbook of Brain Theory and Neural Networks*. The MIT Press, 2002.

[25] Maryam Fazel, Haitham Hindi, and Stephen P. Boyd. A Rank Minimization Heuristic with Application to Minimum Order System Approximation. In *Proc. of the American Control Conference*, 2001.

[26] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with coral. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, San Francisco, California, March 2004.

[27] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, UK, 1995. Springer-Verlag.

[28] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[29] Google TV. `http://www.google.com/tv/`.

[30] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proc. of the ACM/SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[31] William C. Hardy. *Quality of Service for VOIP: Measuring and Evaluating Packet - Switched Services*. McGraw-Hill, Inc., New York, NY, USA, 2002.

[32] Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, Bradley Huffaker, Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, and Bradley Huffaker. On routing asymmetry in the Internet. In *Proc. of IEEE Globecom*, 2005.

[33] Christian Hennig and Mahmut Kutlukaya. Some thoughts about the design of loss functions. *REVSTAT–Statistical Journal*, 5(1), 2007.

[34] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Transactions on Networking*, 11:537–549, August 2003.

[35] Manish Jain and Constantinos Dovrolis. Ten fallacies and pitfalls on end-to-end available bandwidth estimation. In *Proc. of ACM/SIGCOMM Internet Measurement Conference*, 2004.

[36] Shuiwang Ji and Jieping Ye. An accelerated gradient method for trace norm minimization. In *International Conference on Machine Learning*, pages 457–464, 2009.

[37] Qifa Ke and Takeo Kanade. Robust L_1 norm factorization in the presence of outliers and missing data by alternative convex programming. In *Computer Vision and Pattern Recognition*, pages 592–599, 2005.

[38] Raghunandan H. Keshavan, Sewoong Oh, and Andrea Montanari. Matrix completion from a few entries. *CoRR*, abs/0901.3150, 2009.

[39] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[40] J. Ledlie, P. Gardner, and M. I. Seltzer. Network coordinates in the wild. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, Cambridge, April 2007.

[41] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, pages 556–562. MIT Press, 2001.

[42] Kiryung Lee and Yoram Bresler. ADMiRA: Atomic decomposition for minimum rank approximation. *IEEE Transactions on Information Theory*, 56(9):4402–4416, 2010.

[43] S. Lee, Z. Zhang, S. Sahu, and D. Saha. On suitability of Euclidean embedding of Internet hosts. *SIGMETRICS*, 34(1):157–168, 2006.

[44] Y. Liao, W. Du, P. Geurts, and G. Leduc. Decentralized prediction of end-to-end network performance classes. In *Proc. of CoNEXT*, Tokyo, Japan, December 2011.

[45] Y. Liao, W. Du, P. Geurts, and G. Leduc. DMFSGD: A decentralized matrix factorization algorithm for network distance prediction. *CoRR*, abs/1201.1174v1, 2012.

[46] Y. Liao, P. Geurts, and G. Leduc. Network distance prediction based on decentralized matrix factorization. In *Proc. of IFIP Networking Conference*, Chennai, India, May 2010.

[47] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19:2756–2779, Oct 2007.

[48] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the accuracy of embeddings for Internet coordinate systems. In *Proc the IMC Conference*, pages 1–14, New York, NY, USA, 2005. ACM.

[49] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.

[50] Cristian Lumezanu, Dave Levin, and Neil Spring. Peerwise discovery and negotiation of faster paths. In *Proc. ACM HotNets Workshop*, 2007.

[51] Harsha V. Madhyastha, Ethan Katz-bassett, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane nano: Path prediction for peer-to-peer applications. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, Boston, USA, Apr 2009.

[52] Yun Mao, Lawrence Saul, and Jonathan M. Smith. IDES: An Internet distance estimation service for large networks. *IEEE Journal On Selected Areas in Communications*, 24(12):2273–2284, December 2006.

[53] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of Machine Learning Research*, 99:2287–2322, August 2010.

[54] S. Min, J. Holliday, and D. Cho. Optimal super-peer selection for large-scale p2p system. In *Proc. the ICHIT Conference*, pages 588–593, Washington, DC, USA, November 2006.

[55] Animesh Nandi, Aditya Ganjam, Peter Druschel, T. S. Eugene Ng, Ion Stoica, Hui Zhang, and Bobby Bhattacharjee. Saar: a shared control plane for overlay multicast. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2007.

[56] Netflix CineMatch. `http://www.netflix.com/`.

[57] Netflix Prize. `http://www.netflixprize.com/`.

[58] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. of IEEE INFOCOM*, New York, NY, USA, June 2002.

[59] Hu Ningning and Steenkiste Peter. Exploiting internet route sharing for large scale available bandwidth estimation. In *Proc. of ACM/SIGCOMM Internet Measurement Conference*, pages 187–192, Berkeley, CA, USA, 2005. USENIX Association.

[60] Abhinav Pathak, Himabindu Pucha, Ying Zhang, Y. Charlie Hu, and Z. Morley Mao. A measurement study of Internet delay asymmetry. In *Proc. of the Passive and Active Measurement*, Cleveland, OH, USA, April 2008.

[61] Peter Pietzuch, Jonathan Ledlie, Michael Mitzenmacher, and Margo Seltzer. Network-aware overlays with network coordinates. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops*, 2006.

[62] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, Ravi Prasad, and Constantinos Dovrolis Georgia. Bandwidth estimation: Metrics, measurement techniques, and tools. *IEEE Network*, 17:27–35, 2003.

[63] Pavlin Radoslavov, Ramesh Govindan, and Deborah Estrin. Topology-informed Internet replica placement. *Computer Communications*, 25(4):384–392, March 2002.

[64] Venugopalan Ramasubramanian, Dahlia Malkhi, Fabian Kuhn, Mahesh Balakrishnan, Archit Gupta, and Aditya Akella. On the treeness of Internet latency and bandwidth. In *ACM SIGMETRICS / Performance*, Seattle, WA, USA, June 2009.

[65] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, San Diego, CA, USA, August 2001.

[66] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Proc. of IEEE INFOCOM*, June 2002.

[67] Jasson D. M. Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *International Conference on Machine Learning*, pages 713–719, 2005.

[68] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Proc. of the Passive and Active Measurement*, 2003.

[69] A. Rowstron and P. Drusche. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM ICDSP*, Heidelberg, Germany, 2001.

[70] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. of the International World Wide Web Conference - WWW10*, Hong Kong, May 2001.

[71] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. In *Proc. of ACM SIGCOMM*, Cambridge, MA, USA, September 1999.

[72] Rahul Shah, Ravi Jain, and Farooq Anjum. Efficient dissemination of personalized information using content-based multicast. In *Proc. of IEEE INFOCOM*, New York, NY, USA, June 2002.

[73] Shai Shalev-Shwartz, Alon Gonen, and Ohad Shamir. Large-Scale Convex Minimization with a Low-Rank Constraint. In *International Conference on Machine Learning*, 2011.

[74] Alok Shriram, Margaret Murray, Young Hyun, Nevil Brownlee, Andre Broido, Marina Fomenkov, and Kc Claffy. Comparison of public end-to-end bandwidth estimation tools on high-speed links. In *Proc. of the Passive and Active Measurement*, pages 306–320, 2005.

[75] Han Hee Song, Lili Qiu, Senior Member, Yin Zhang, and Senior Member. NetQuest: A flexible framework for large-scale network measurement. In *In Proc. of ACM SIGMETRICS*, 2006.

[76] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. Enabling efficient content location and retrieval in peer-to-peer systems by exploiting locality in interests. *ACM Computer Communication Review*, 32(1):80–80, 2002. Abstract of poster in ACM SIGCOMM 2001.

[77] James Surowiecki. *The Wisdom of Crowds*. Anchor, 2005.

[78] Michał Szymaniak, David Presotto, Guillaume Pierre, and Maarten van Steen. Practical large-scale latency estimation. *Elsevier Computer Networks*, 52(7):1343–1364.

[79] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, June 2009.

[80] Liying Tang and Mark Crovella. Virtual landmarks for the Internet. In *Proc. of ACM/SIGCOMM Internet Measurement Conference*, October 2003.

[81] Xueyan Tang and Jianliang Xu. Qos-aware replica placement for content distribution. *IEEE Trans. Parallel Distrib. Syst.*, 16(10):921–932, October 2005.

[82] Vuze Bittorrent. `http://www.vuze.com/`.

[83] G. Wang, B. Zhang, and T. S. E. Ng. Towards network triangle inequality violation aware distributed systems. In *Proc. the ACM/IMC Conference*, pages 175–188, San Diego, CA, USA, October 2007.

[84] Guohui Wang and T. S. Eugene Ng. Distributed algorithms for stable and secure network coordinates. In *Proc. of ACM/SIGCOMM Internet Measurement Conference*, Vouliagmeni, Greece, October 2008.

[85] Tim Wauters, Jan Coppens, Bart Dhoedt, and Piet Demeester. Load balancing through efficient distributed content placement. In *2005 Next generation Internet networks*, pages 99–105, 2005.

[86] Zaiwen Wen, Wotao Yin, and Yin Zhang. Solving a low-rank factorization model for matrix completion by a non-linear successive over-relaxation algorithm. Technical Report TR10-07, Department of Computational and Applied Mathematics, Rice University, 2010.

[87] B. Wong, A. Slivkins, and E. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proc. of ACM SIGCOMM*, August 2005.

[88] M. Wu. Collaborative filtering via ensembles of matrix factorizations. In *KDD Cup and Workshop at the 13th ACM SIGKDD Conference*, 2007.

[89] Praveen Yalagandula, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Sung-Ju Lee. S3: a scalable sensing service for monitoring large networked systems. In *Proc. of the 2006 SIGCOMM workshop on Internet network management*, pages 71–76, 2006.

[90] Rongmei Zhang, Chunqiang Tang, Y. Charlie Hu, Sonia Fahmy, and Xiaojun Lin. Impact of the inaccuracy of distance prediction algorithms on Internet applications: an analytical and comparative study. In *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[91] H. Zheng, E. K. Lua, M. Pias, and T. Griffin. Internet Routing Policies and Round-Trip-Times. In *Proc. of the Passive and Active Measurement*, Boston, MA, USA, April 2005.