DMFSGD: A Decentralized Matrix Factorization Algorithm for Network Distance Prediction

Yongjun Liao, Wei Du, Pierre Geurts and Guy Leduc

***Abstract*—The knowledge of end-to-end network distances is essential to many Internet applications. As active probing of all pairwise distances is infeasible in large-scale networks, a natural idea is to measure a few pairs and to predict the other ones without actually measuring them. This paper formulates the prediction problem as matrix completion where the unknown entries in a pairwise distance matrix constructed from a network are to be predicted. By assuming that the distance matrix has a low-rank characteristics, the problem is solvable by low-rank approximation based on matrix factorization. The new formulation circumvents the well-known drawbacks of existing approaches based on Euclidean embedding.**

A new algorithm, so-called Decentralized Matrix Factorization by Stochastic Gradient Descent (DMFSGD), is proposed. By letting network nodes exchange messages with each other, the algorithm is fully decentralized and only requires each node to collect and to process local measurements, with neither explicit matrix constructions nor special nodes such as landmarks and central servers. In addition, we compared comprehensively matrix factorization and Euclidean embedding to demonstrate the suitability of the former on network distance prediction. We further studied the incorporation of a robust loss function and of non-negativity constraints. Extensive experiments on various publicly-available datasets of network delays show not only the scalability and the accuracy of our approach, but also its usability in real Internet applications.

***Index Terms*—network distance prediction, matrix completion, matrix factorization, stochastic gradient descent.**

I. Introduction

On large networks such as the Internet, many applications require the knowledge of end-to-end network distances in order to achieve Quality of Service (QoS) objectives. In the networking field, the distance between two network nodes is defined by the delay or latency between them, in the form of either one-way delay or more often round-trip time (RTT). Examples include peer-to-peer file sharing, overlay networks, and content distribution systems where peers preferably access nodes or servers that are likely to respond fast [1]–[5].

Clearly, it is infeasible to probe actively end-to-end distances among all pairs of nodes in large networks as the demand in terms of measurements grows quadratically with the scale of the network. A natural idea is to probe a small set of pairs and then predict the distances between other pairs where there are no direct measurements. This understanding has motivated numerous research on Network Coordinate System (NCS) [6]–[10]. For instance, approaches based on Euclidean embedding have been widely studied and achieved good performance in interesting scenarios [7], [11]. Realizing that the assumption of Euclidean distance properties (symmetry and triangle inequality) are often violated in practice, as observed in various studies [7], [12]–[16], matrix factorization has recently drawn increasing attention of the networking community [8], [9].

In this paper, we investigate matrix factorization for network distance prediction. In particular, we formulate the problem of network distance prediction as a matrix completion problem where a partially observed matrix is to be completed [17]–[19]. Here, the matrix contains distance measurements such as RTTs between network nodes with some of them known and the others unknown, thus to be filled. Matrix completion requires the matrix being completed to have a low-rank characteristics, which has been consistently observed in network distance matrices extracted from various datasets including ours [20]–[22]. Although numerous approaches to matrix completion have been proposed, many of which are based on low-rank matrix factorization [23]–[25], very few are directly applicable to network applications where decentralized processing of data is most of the time a necessity. In this paper, we propose a fully decentralized algorithm based on Stochastic Gradient Descent (SGD), which is founded on the stochastic optimization theory with nice convergence guarantees [26].

The so-called Decentralized Matrix Factorization by Stochastic Gradient Descent (DMFSGD) algorithm has two distinct features. First, 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 number of distance measurements. Second, the algorithm is simple, with no infrastructure, and is computationally lightweight, containing only vector operations. These features make it 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. Extensive experiments on various publicly-available RTT datasets show not only the scalability and the accuracy of our approach but also its usability in real Internet applications.

Our preliminary work on decentralized matrix factorization for network distance prediction was published in [9]. In the

Yongjun Liao and Guy Leduc are with Research Unit in Networking (RUN), University of Liège, Belgium. Email: {yongjun.liao, guy.leduc}@ulg.ac.be.

Wei Du is a visiting researcher at Research Unit in Networking (RUN), University of Liège, Belgium. Email: weidu@montefiore.ulg.ac.be. This work was done when he was a postdoctoral researcher at Intelligent and Interactive Systems (IIS), University of Innsbruck, Austria.

Pierre Geurts is with Systems and Modeling, University of Liège, Belgium. Email: p.geurts@ulg.ac.be.

present paper, we make the following distinct contributions:

- Our previous approach in [9] was based on Alternating Least Squares (ALS), requiring each node to probe all local measurements simultaneously. In contrast, the new SGD-based approach allows each node to probe one measurement at a time, making the system more flexible. The new approach also addresses several issues arising when applied practically, including the difficult choice of the learning rate parameter and the consideration of passive distance acquisitions and dynamic measurements.
- We compare comprehensively matrix factorization and Euclidean embedding to reveal the suitability of matrix factorization. A unified view is provided which leads to a unified optimization framework to solve both of them.
- Two extensions of the current matrix factorization model are proposed, including the incorporation of a robust loss function and the introduction of constraints in the model to ensure the nonnegativity of the predicted distances. These extensions are found helpful in improving the accuracy of the prediction and require little modification to the algorithm with no additional computational cost.
- In addition, more extensive evaluations have been carried out to study not only the impact of the parameters but also the accuracy of our approach. In particular, we highlight the usability of our approach by simulations on real dynamic data collected from a peer-to-peer file sharing application, namely Azureus [11], [27].

The rest of the paper is organized as follows. Section II summarizes the related work on network distance prediction. Section III introduces the formulation of network performance prediction as matrix completion and its resolution by low-rank matrix factorization. Section IV describes the decentralized matrix factorization algorithm based on Stochastic Gradient Descent. Section V discusses possible extensions to the current matrix factorization model. Section VI provides a unified view of different approaches to network distance prediction. Section VII evaluates our approach on various RTT datasets. Conclusions and future work are given in Section VIII.

II. Related Work

Among numerous works on network distance prediction, we only discuss and compare approaches based on Euclidean embedding and on matrix factorization due to their simplicity and generality. We refer the interested readers to [10] for a more detailed review of this field.

A. Euclidean Embedding

A straightforward approach to network distance prediction is to embed network nodes into a metric space, commonly the Euclidean coordinate system, where each node is assigned a coordinate from which distances can be directly computed. Figure 1 illustrates Euclidean embedding for network distance prediction. Two representative approaches are Global Network Positioning (GNP) [6] and Vivaldi [7].

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 an Euclidean coordinate system. 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 spring and minimizes its energy according to Hooke's law to find an optimal embedding.

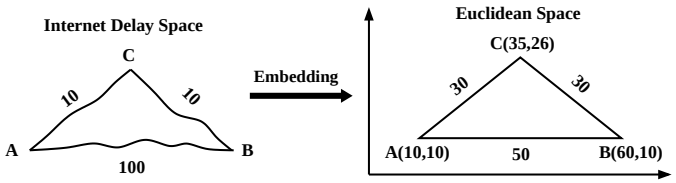


Fig. 1. Network distance prediction by Euclidean embedding.

In a metric space, 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 [28], [29]. 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 [7], [12]–[16]. In the presence of TIVs, metric space embedding shrinks the long edges and stretches the short ones, degrading heavily the accuracy of the embedding.

B. Matrix Factorization

Alternatively, matrix factorization has also been used for network distance prediction (see Figure 2 for an illustration). The biggest advantage of matrix factorization is that it makes no assumption of Euclidean distance properties and thus can tolerate the widespread TIVs and the possible asymmetry of network distance measurements.

The first system based on matrix factorization was Internet Distance Estimation Service (IDES) [8] 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). Landmark-based systems suffer from several drawbacks including single-point failures, landmark overloads and potential security problems. Like Vivaldi, our previous work DMF extended IDES by eliminating the landmarks and solved the matrix factorization problem by ALS [9].

III. Network Distance Prediction by Low-Rank Matrix Factorization

A. Problem Formulation

Assuming n nodes in the network, a $n \times n$ distance matrix is constructed, denoted by D with d_{ij} the measured distance from node i to node j. D is largely incomplete, as only a few distances between pairs of nodes are measured. Let $\hat{D}$ be the predicted matrix with $\hat{d}_{ij}$ the predicted distance computed from some function.

Given the above notations, we formulate the problem of network distance prediction as a matrix completion problem

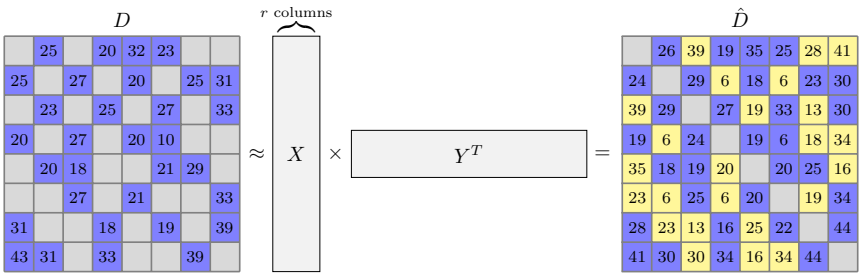


Fig. 2. Network distance prediction by matrix factorization. Note that the diagonal entries of D and $\hat{D}$ are empty.

that estimates the missing entries in D from a small number of known entries. Although relatively new, matrix completion has been well studied and understood [17]–[19]. The basic requirement is that D has to be low-rank or approximately low-rank. That is, D can only have r significant singular values, where $r \ll n$, and the others either vanish or are negligible. Under this and some other suitable conditions, with high probability, D can be recovered exactly or accurately from just $O(n^b r \log n)$ randomly observed entries, where b is only slightly larger than 1. Such result is interesting, because it shows that recovering a low-rank matrix from far less than n^2 entries is really possible.

Generally, matrix completion is solved by minimizing the following objective function

$$L(D, \hat{D}, W) = \sum_{i,j=1}^{n} w_{ij} l(d_{ij}, \hat{d}_{ij}), \quad \text{subject to } \mathit{Rank}(\hat{D}) \leqslant r \tag{1}$$

where W is a weight matrix with w_{ij} taking values between 0 and 1. In a simple case, $w_{ij} = 1$ if d_{ij} is observed and 0 otherwise. Note that if the distance measurements are RTTs, then $d_{ji} = d_{ij}$ as RTTs are approximately symmetric. Consequently, $w_{ji} = w_{ij}$ as d_{ji} and d_{ij} are either both known or both unknown. l is a loss function that penalizes the difference between an estimate and its true value. The L_2 or square loss function is the most commonly-used, given by

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

We will discuss other loss functions in Section V.

B. Low-Rank Characteristics

To apply matrix completion, the low-rank requirement has to be met. We empirically evaluate it for two RTT matrices by the spectral plots in Figure 3. It can be seen that the singular values of both matrices decrease fast as the 10th singular values are 5.7% and 2.9% of the largest ones respectively.

The low-rank characteristics has been consistently observed in matrices extracted from various measurement datasets and studied in depth [20]–[22]. It roots solidly in the redundancy of link usage across paths because Internet paths with nearby end nodes often overlap or share bottleneck links. This causes distance measurements on different paths to be correlated and induces the related matrix to be approximately low-rank. Thus, although only empirically justified, these insights suggest that the low-rank phenomenon is likely to be general and exists in a wide variety of scenarios.

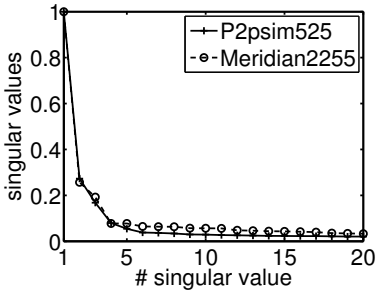


Fig. 3. The singular values of a RTT matrix of 2255×2255, extracted from the Meridian dataset [30] and called "Meridian2255", and of a RTT matrix of 525×525, extracted from the P2PSim dataset [30] and called "P2PSim525". The singular values are normalized so that the largest singular values of both matrices are equal to 1.

C. Low-Rank Matrix Factorization

Directly finding $\hat{D}$ by minimizing eq. 1 is difficult due to the rank constraint. However, as $\hat{D}$ is of low rank, we can factorize it into the product of two smaller matrices, i.e.,

$$\hat{D} = XY^T, \tag{3}$$

where X and Y are of size $n \times r$. Therefore, we can get rid of the rank constraint by replacing $\hat{D}$ by XY^T in eq. 1, and then look for X and Y instead by minimizing

$$L(D, X, Y, W) = \sum_{i,j=1}^{n} w_{ij} l(d_{ij}, x_i y_j^T), \tag{4}$$

where x_i is the ith row of X, y_i is the ith row of Y, and $x_i y_j^T = \hat{d}_{ij}$ is the estimate of d_{ij}. Note that the factorization in eq. 3 has no unique solution as

$$\hat{D} = XY^T = XGG^{-1}Y^T, \tag{5}$$

where G is any arbitrary $r \times r$ invertible matrix. Thus, replacing X by XG and Y^T by $G^{-1}Y^T$ results in the same $\hat{D}$.

When D is complete, analytic solutions to the optimization of eq. 4 can be found by using singular value decomposition (SVD) [31]. With missing entries, matrix factorization is usually done by iterative optimization methods such as Gradient Descent or Newton algorithms [24]. Note that additional constraints can be imposed in eq. 4. For instance, the entries of X and Y can be required to be nonnegative in order

to recover a nonnegative matrix, leading to the nonnegative matrix factorization (NMF) [32].

D. Incorporation of the Regularization

Matrix completion by matrix factorization suffers from a well-known problem called overfitting in the field of machine learning [33]. In words, directly optimizing eq. 4 often leads to a "perfect" model with no or small errors on the training data while having large errors on the unseen data which are not used in learning. The problem is more severe when D is sparse or when r is large.

A common way to avoid overfitting is through regularization that penalizes the norms of the solutions, resulting in the following regularized objective function,

$$L(D,X,Y,W,\lambda) = \sum_{i,j=1}^{n} w_{ij} l(d_{ij}, x_i y_j^T) + \lambda \sum_{i=1}^{n} x_i x_i^T + \lambda \sum_{i=1}^{n} y_i y_i^T, \quad (6)$$

where λ is the regularization coefficient that controls the extent of regularization. The regularization also avoids the numerical instability of the solutions due to the non-uniqueness of the factorization (see eq. 5). Among the infinite number of pairs of X and Y which produce the same $\hat{D}$, the regularization forces to choose the pair with the smallest norm.

IV. Decentralized Matrix Factorization for Network Distance Prediction

The goal is to find X and Y such that XY^T best approximates D by optimizing Eq. 6. This section introduces our approach to achieving this goal in a fully decentralized manner.

A. Problem Formulation

The decentralized optimization of eq. 6 forbids the explicit constructions of matrices D, X and Y. Thus, available measurements in D are probed and processed locally at each node. As distance measurements are available at the sender nodes, we let the senders process the acquired measurements. We also have the rows of X and of Y distributively stored, i.e., x_i and y_i are stored at node i. In what follows, x_i and y_i are called the x and y *coordinates* of node i.

To calculate x_i and y_i, node i collaborates with a number of nodes in the network, called *neighbors* in what follows, and optimizes the following losses

$$l_i = \sum_{j=1}^{n} w_j^i l(d_{ij}, x_i y_j^T) + \lambda x_i x_i^T, \quad (7)$$

$$l^i = \sum_{j=1}^{n} w_j^i l(d_{ji}, x_j y_i^T) + \lambda y_i y_i^T, \quad (8)$$

where w_j^i represents the neighboring relationship of node j to node i, i.e., $w_j^i = 1$ if node j is a neighbor of node i or equivalently if node i probes node j, and 0 otherwise. Note that as w_j^i is not necessarily equal to w_i^j, measurements d_{ij} and d_{ji} may only be known by either node i or node j but not by the other.

Essentially, l_i is the regularized loss of the edges from node i to other nodes and l^i is that of the edges from other nodes to node i. Thus, we address the large-scale optimization problem in eq. 6 by decomposing it into a number of subproblems in eqs. 7 and 8 which can be solved locally at each node.

In seeing that eqs. 7 and 8 are standard least-squares problems where analytical solutions exist, our previous work in [9] solved the matrix factorization problem by Alternating Least Squares (ALS), which alternatively and iteratively solves the small least-squares problems in eqs. 7 and 8. While the ALS-based algorithm performed well in simulations on datasets containing static measurements, it requires each node to probe measurements with a number of nodes simultaneously, which is impractical when deployed in real applications. Below, we propose a different algorithm based on Stochastic Gradient Descent (SGD) whereby each node can process one measurement at a time.

B. Stochastic Gradient Descent (SGD)

SGD is a variation of traditional Batch Gradient Descent which is often used for online machine learning [26]. 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. SGD is particularly appropriate for network applications, as 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.

1) Stochastic Updates: When using SGD, each node probes one neighbor at a time, measures its distance with respect to that node and retrieves that node's coordinates. Let node j be the neighbor chosen by node i at the current time. Then, the regularized losses that node i seeks to reduce with respect to node j are

$$l_{ij} = l(d_{ij}, x_i y_j^T) + \lambda x_i {x_i}^T, \quad (9)$$

$$l_{ji} = l(d_{ji}, x_j y_i^T) + \lambda y_i {y_i}^T. \quad (10)$$

The gradients of l_{ij} and l_{ji} are

$$\frac{\partial l_{ij}}{\partial x_i} = \frac{\partial l(d_{ij}, x_i y_j^T)}{\partial x_i} + \lambda x_i, \quad (11)$$

$$\frac{\partial l_{ji}}{\partial y_i} = \frac{\partial l(d_{ji}, x_j y_i^T)}{\partial y_i} + \lambda y_i. \quad (12)$$

In particular, the gradients of the L_2 loss function are

$$\frac{\partial l}{\partial x_i} = -(d_{ij} - x_i y_j^T) y_j, \quad (13)$$

$$\frac{\partial l}{\partial y_i} = -(d_{ji} - x_j y_i^T) x_j. \quad (14)$$

Note that we dropped the factor 2 from the derivatives of the regularization terms and of the L_2 loss function for mathematical convenience.

Algorithm 1 Line Search (for updating x_i)

1: compute l_i^0 by eq. 7;
2: initialize η with a large value;
3: **for** $i = 1$ to $maxNumberLineSearch$ **do**
4: compute x_i by eq. 17;
5: compute l_i by eq. 7;
6: **if** $l_i < l_i^0 + \delta$ **then**
7: **return**
8: **end if**
9: $\eta \longleftarrow \eta/2$;
10: **end for**

Then, node i updates its coordinates along the negative gradient directions, given by

$$x_i = (1-\eta\lambda)x_i + \eta(d_{ij} - x_i y_j^T)y_j, \quad (15)$$

$$y_i = (1-\eta\lambda)y_i + \eta(d_{ji} - x_j y_i^T)x_j, \quad (16)$$

where η, called *learning rate* or *step size*, controls the speed of the updates.

2) Minibatch and Line Search: The SGD algorithm is sensitive to the learning rate η, where a too large η results in large steps of updates and may overflow the solution, whereas a too small η makes the convergence slow. This sensitivity can be relieved by using more training samples at the same time, leading to minibatch SGD with the following update rules

$$x_i = (1-\eta\lambda)x_i + \eta\sum_{j=1}^{n} w_j^i(d_{ij} - x_i y_j^T)y_j, \quad (17)$$

$$y_i = (1-\eta\lambda)y_i + \eta\sum_{j=1}^{n} w_j^i(d_{ji} - x_j y_i^T)x_j. \quad (18)$$

To completely get rid of η, a line search strategy can be incorporated to determine η adaptively [34]. In particular, in each update, η starts with a large initial value and is gradually decreased until the losses in eqs. 7 or 8 are reduced after the update. The line search algorithm for updating x_i is given in Algorithm 1. The same algorithm can be used for updating y_i by replacing eq. 7 by eq. 8 and eq. 17 by eq. 18. Note that δ in Line 6 is a small positive constant that helps overcome poor local optimums. We will demonstrate the effectiveness of adapting η by line search using Algorithm 1 in Section VII.

C. Neighbor Decay and Neighbor Selection

As mentioned earlier, it is preferable to have a system that probes and processes measurements one by one. Thus, we let each node maintain the information (distance measurements and coordinates) of its neighbors, i.e., the nodes with which it communicates. In minibatch SGD, each node probes one neighbor at a time but updates its coordinates with respect to all neighbors in the neighbor set using their 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.,

$$w_j^i = \frac{a_{max} - a_j}{\sum_{j \in \mathrm{NeighborSet}(i)}(a_{max} - a_j)}, \quad (19)$$

Algorithm 2 DMFSGD(i, j)

1: node i retrieves d_{ij}, d_{ji}, x_j, y_j actively or passively;
2: node i updates the weights of its neighbors by eq. 19;
3: update x_i by eq. 17 with η set by line search;
4: update y_i by eq. 18 with η set by line search;

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. Note that this neighbor decay strategy was firstly proposed by [11] to overcome the problem of skewed neighbor update in Vivaldi. In words, some nodes may stay in the system for much longer time than others and they will be probed at far greater frequency. A direct consequence is that the optimization will become skewed toward these nodes.

Conventionally, the neighbors of a node are selected randomly and the distances between a node and its neighbors are probed by active measurements [7]. However, in practice, it is more attractive to perform the updates of the coordinates passively without generating any extra traffic. In some applications such as Azureus, passivity is enforced, as we have no control over the selection of neighbors with which a node communicates and when it communicates with them [11].

Therefore, we differentiate the situations where 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 15 and 16 or in eqs 17 and 18.

D. Algorithm

We denote the SGD-based decentralized matrix factorization algorithm as DMFSGD, given in Algorithm 2. The algorithm has no infrastructure and employs the same process at all nodes. It is simple, with update rules containing only vector operations. Unlike Vivaldi which is limited to RTTs [7], DMFSGD can also deal with other distance measurements such as one-way delays which are asymmetric. For RTTs, node i measures d_{ij} and assumes $d_{ji} = d_{ij}$, whereas for one-way delays, node i has to measure both d_{ij} and d_{ji}. In the implementation, the coordinates of each node are initialized with random numbers uniformly distributed between 0 and 1. Empirically, the algorithm is insensitive to the random initializations of the coordinates.

V. Extended Matrix Factorization Models

This section discusses two possible ways to extend the common matrix factorization model.

A. Robust Matrix Factorization

The widely-used L_2 loss function is known to be sensitive to 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 [35], [36]. For example, the L_1 loss function is defined as

$$l(d, \hat{d}) = |d - \hat{d}|. \tag{20}$$

Thus, we can potentially enhance the robustness of matrix factorization 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 [1], given by

$$\frac{\partial l}{\partial x_i} = -\text{sign}(d_{ij} - x_i y_j^T) y_j, \tag{21}$$

$$\frac{\partial l}{\partial y_i} = -\text{sign}(d_{ji} - x_j y_i^T) x_j. \tag{22}$$

Replacing the gradient functions of eqs. 11 and 12 by eqs. 21 and 22, the update rules of minibatch SGD become

$$x_i = (1 - \eta\lambda) x_i + \eta \sum_{j=1}^{n} \text{sign}(d_{ij} - x_i y_j^T) w_j^i y_j, \tag{23}$$

$$y_i = (1 - \eta\lambda) y_i + \eta \sum_{j=1}^{n} \text{sign}(d_{ji} - x_j y_i^T) w_j^i x_j, \tag{24}$$

Comparing eqs. 23 and 24 with eqs. 17 and 18, the only difference is that for the L_2 loss function, the magnitudes of the updates are proportional to the fitting errors $(d - xy^T)$, 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.

B. NonNegativity Constraint

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

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

$$\sum_{i,j=1}^{n} w_{ij} l(d_{ij}, x_i y_j^T) + \lambda \sum_{i=1}^{n} x_i x_i^T + \lambda \sum_{i=1}^{n} y_i y_i^T, \tag{25}$$

$$\text{subject to } x_i \geqslant 0,\ y_i \geqslant 0,\ i = 1, \ldots, n.$$

The optimization of NMF is not fundamentally different from that of the unconstrained matrix factorization, adding only one projection step that turns the negative entries in x_i and y_i into zero after each SGD update which causes no noticeable impact on the speed of the algorithm. The technique is also known as projected gradient descent [37].

[1]Analogously, the subgradient-based technique that optimizes non-differentiable functions is called subgradient descent [34]. Following the convention in [26], we use the term SGD to refer to both Stochastic Gradient and SubGradient Descent.

Algorithm 3 Extended_DMFSGD(i, j)

```
node i retrieves d_ij, d_ji, x_j, y_j actively or passively;
node i updates the weights of its neighbors by eq. 19;
if use L_2 loss function then
    update x_i by eq. 17 with η set by line search;
    update y_i by eq. 18 with η set by line search;
else // use L_1 loss function
    update x_i by eq. 23 with η set by line search;
    update y_i by eq. 24 with η set by line search;
end if
if force nonnegativity then
    turn the negative entries in x_i and y_i into 0;
end if
```

Note that the nonnegativity constraint has been previously studied in [8], which adopted a more heavyweight nonnegative least-squares solver.

C. Extended DMFSGD Algorithm

Empirically, we found that the incorporation of the nonnegativity constraint and the robust loss function not only improved the accuracy but also made the results more stable and less sensitive to parameter settings, which will be demonstrated in Section VII. The extended DMFSGD algorithm is given in Algorithm 3. Note that as the basic version in Algorithm 2 is a special case of the extended version in Algorithm 3, we will refer to Algorithm 3 simply as DMFSGD in what follows.

VI. A Unified View of Network distance prediction

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

A. A Unified Formulation

Various approaches to network distance prediction have adopted different models, based on Euclidean embedding or matrix factorization, and different architectures, with landmarks or without. Nevertheless, these seemingly different approaches all optimize an objective function of the following form:

$$L(D, \hat{D}, W) = \sum_{i,j=1}^{n} w_{ij} l(d_{ij}, \hat{d}_{ij}). \tag{26}$$

They differ only in the setting of w_{ij} in W and in the associated distance function to calculate $\hat{d}_{ij}$.

1) The Setting of w_{ij}: For landmark-based methods, as all paths between landmarks are measured and ordinary nodes probe only the landmarks,

$$w_{ij} = \begin{cases} 1 & \text{if node } j \text{ is a landmark} \\ 0 & \text{otherwise} \end{cases}.$$

For decentralized methods that rely on no landmark, as each node equally probes a number of neighbors,

$$w_{ij} = \begin{cases} 1 & \text{if node } i \text{ probes node } j \\ 0 & \text{otherwise} \end{cases}.$$

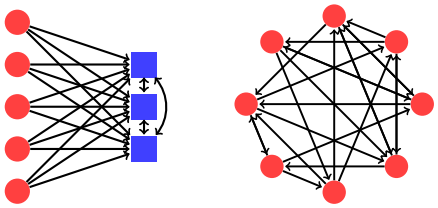


Fig. 4. Architectures of landmark-based, the left plot, and decentralized, the right plot, systems for network distance prediction. The squares are landmarks and the circles are ordinary nodes. The directed path from node i to node j means that node i probes node j and therefore $w_{ij} = 1$.

Figure 4 illustrates the architectures of landmark-based and decentralized systems.

2) Distance functions to calculate $\hat{d}_{ij}$: For matrix factorization,

$$\hat{d}_{ij} = x_i y_j^T, \tag{27}$$

For Euclidean embedding,

$$\hat{d}_{ij} = \sqrt{(x_i - x_j)^T (x_i - x_j)}, \tag{28}$$

which is the Euclidean distance. x_i is the Euclidean coordinate of node i.

The different choices of the distance function naturally impose different constraints on the produced results. For instance, the use of the Euclidean distance forces $\hat{D}$ to be symmetric and TIV-free, while the use of our distance forces $\hat{D}$ to be low-rank. Table I compares the main features of matrix factorization and Euclidean embedding.

B. 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 our 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 our distance in eq 27 by the Euclidean distance in eq. 28, which leads essentially to the update rule of Vivaldi [7], given by

$$x_i = x_i - \eta \frac{\partial l(d_{ij}, \hat{d}_{ij})}{\partial x_i} = x_i + \eta (d_{ij} - \hat{d}_{ij}) \frac{x_i - x_j}{\hat{d}_{ij}}.$$

Note that 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 [7].

VII. Experiments and Evaluations

In this section, we evaluate our DMFSGD algorithm[2] and compare it with state-of-the-art approaches.

[2]The matlab implementation of the algorithm is available at http://www.run.montefiore.ulg.ac.be/~liao/DMFSGD.

A. Evaluation Methodology

The evaluations were performed under the following criteria and on the following datasets.

1) Evaluation Criteria:

- **Cumulative Distribution of Relative Estimation Error** Relative Estimation Error (REE) is defined as

$$REE = \frac{|\hat{d}_{ij} - d_{ij}|}{d_{ij}}.$$

- **Stress** Stress measures the overall fitness and is used to illustrate the convergence of the algorithm, defined as

$$stress = \sqrt{\frac{\sum_{i,j=1}^{n} (d_{ij} - \hat{d}_{ij})^2}{\sum_{i,j=1}^{n} {d_{ij}}^2}}.$$

- **Median Absolute Error** Median Absolute Error (MAE) is defined as

$$MAE = median_{ij}(|d_{ij} - \hat{d}_{ij}|).$$

2) Datasets:

- **Harvard226** contains dynamic and passive measurements of application-level RTTs, with timestamps, between 226 Azureus clients collected in 4 hours [11].
- **P2PSim1740** was obtained from the P2PSim project that contains static RTT measurements between 1740 Internet DNS servers [38].
- **Meridian2500** was obtained from the Cornell Meridian project that contains static RTT measurements between 2500 nodes [30].
- **P2PSim525** is a complete submatrix between 525 nodes derived from P2PSim1740.
- **Meridian2255** is a complete submatrix between 2255 nodes derived from Meridian2500.
- **Synthetic1000** 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 were only used in section VII-B for the purpose of comparing the models of Euclidean embedding and matrix factorization.

Table II summarizes these datasets. Note that we can neither tell the symmetry nor calculate the TIV percentage of the Harvard226 dataset, as the measurements between network nodes vary over time largely, sometimes in several orders of magnitudes. The Harvard226 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.

3) Implementations for Different Datasets: As mentioned earlier, the DMFSGD algorithm adopts the conventional random neighbor selection procedure in the scenarios where measurements are probed actively and maintains dynamically an

TABLE I
MATRIX FACTORIZATION VS. EUCLIDEAN EMBEDDING

	Matrix Factorization	Euclidean Embedding
optimization objective	$\sum w_{ij} l(d_{ij}, \hat{d}_{ij})$	$\sum w_{ij} l(d_{ij}, \hat{d}_{ij})$
distance function	$\hat{d}_{ij} = x_i y_j^T$	$\hat{d}_{ij} = \sqrt{(x_i - x_j)^T (x_i - x_j)}$
node coordinate	$x_i = (x_{i1}, \cdots, x_{ir})$ $y_i = (y_{i1}, \cdots, y_{ir})$	$x_i = (x_{i1}, \cdots, x_{ir})$
constraints	low rank	Symmetry: $\hat{d}_{ij} = \hat{d}_{ji}$ Triangle Inequality: $\hat{d}_{ij} < \hat{d}_{ik} + d_{kj}$

TABLE II
PROPERTIES OF THE DATASETS

Dataset	Nodes	Symmetry	TIV percentage	Dynamic
Harvard226	226	/	/	Yes
P2PSim1740	1740	Yes	85.53%	No
Meridian2500	2500	Yes	96.55%	No
P2PSim525	525	Yes	76.17%	No
Meridian2255	2255	Yes	96.25%	No
Synthetic1000	1000	Yes	No	No

active neighbor set for each node in the scenarios where measurements are obtained passively. Thus, for the Harvard226 dataset, we let each node maintain 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 3) as they are static.

To handle the dynamics of the measurements in Harvard226, the distance filter in [11] 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, we built a static distance matrix by extracting the median values of the streams of measurements between each pair of nodes and used it as the ground truth.

B. Euclidean Embedding vs. Matrix Factorization

We first compare Euclidean embedding and matrix factorization, to highlight the suitability of the low-rank constraint over the Euclidean distance constraints on the modeling of the network delay space.

1) Algorithms: As the goal is to compare the model suitability, we should ideally 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 [31]. Generally, SVD factorizes D into three matrices, i.e.,

$$D = USV^T,$$

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 D.

To obtain a low-rank factorization, we keep only the r largest singular values in S and replace the other small ones by zero. Let S_r be the new S, $X = US_r^{\frac{1}{2}}$ and $Y^T = S_r^{\frac{1}{2}} V^T$, where $S_r(i,i)^{\frac{1}{2}} = \sqrt{S_r(i,i)}$. Then, $\hat{D} = XY^T$ is the globally optimal low-rank approximation to D.

For Euclidean embedding, also known as MultiDimentional Scaling (MDS), no algorithm can find a global optimum because this problem is non-convex [39]. Nevertheless, the MDS implementation in matlab, mdscale [40], has been widely used and was found to work well on our datasets. To reduce the chance of being trapped in a poor local optimum, we ran mdscale for multiple times with different random initializations and found similar solutions in every run, which gives us good confidence in the results achieved.

2) Evaluations: The model comparison was performed on the three complete datasets including Synthetic1000, P2PSim525 and Meridian2255. On each dataset, we ran MDS and SVD in different dimensions and ranks and computed the stresses and MAE, shown in figure 5. 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 Synthetic1000 which is TIV-free. On P2PSim525 and Meridian2255 where severe TIVs exist, MDS achieves no or little visible improvement beyond 10 dimensions.

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. In Section VII-C, we 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 distance measurements, the low-rank constraint in matrix factorization is more suitable for modeling the network delay space on the Internet. In Section VII-D, we will show that this advantage takes effect

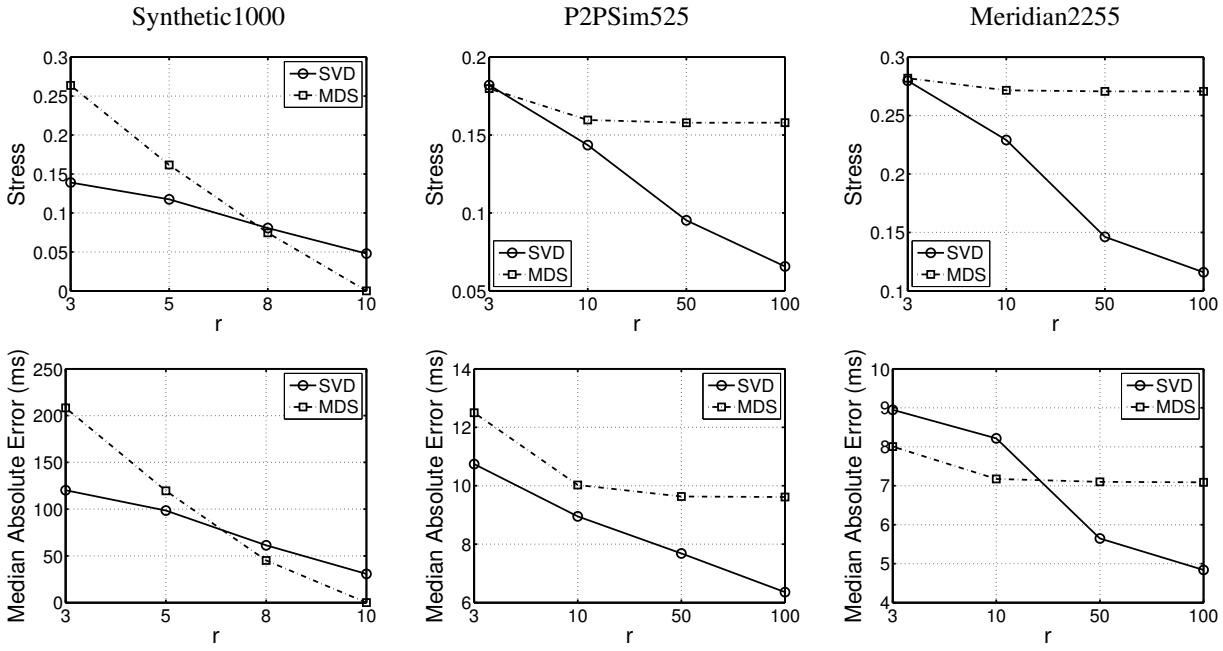


Fig. 5. Comparison of MDS-based Euclidean embedding and SVD-based matrix factorization on synthetic1000, P2PSim525 and Meridian2255. The stresses and the median absolute errors by both methods in different dimensions/ranks are shown on the first and second rows respectively. Note that a perfect embedding with no errors was generated for Synthetic1000 in the 10 dimensional Euclidean space by MDS.

when dealing with real incomplete network distance matrices using our DMFSGD algorithm.

C. Impact of Parameters

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

1) 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 [7], we set $k = 32$ by default for P2PSim1740 and Meridian2500. Note that $k = 32$ makes the available measurements considerably sparse. For instance, $32/1740 = 1.84\%$ measurements are available for each node in P2PSim1740 and $32/2500 = 1.28\%$ for each node in Meridian2500.

2) rank r: Given a delay matrix D, 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 D are abandoned. On the other hand, recovering D 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, we answer this question empirically for our datasets.

3) regularization λ: λ controls the overfitting and improves the numerical stability of the solutions.

4) learning rate η: As mentioned earlier, SGD is sensitive to η where a too large η leads to numerical overflows and a too small η slows down the convergence. Thus, we adapt η 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.

5) Experiments and Observations: We first experimented with different configurations of $r = \{3, 10, 100\}$ and $\lambda = \{0.01, 0.1, 1, 10\}$, with different loss functions and whether to incorporate the nonnegativity constraint. Results are shown in Figure 6. In this experiment, η is adapted by the line search.

We made the following observations. 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 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 Harvard226 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 we often have limited available measurements, a relatively small value of $r = 10$ is adopted by default.

We then experimented with different constant η's and with

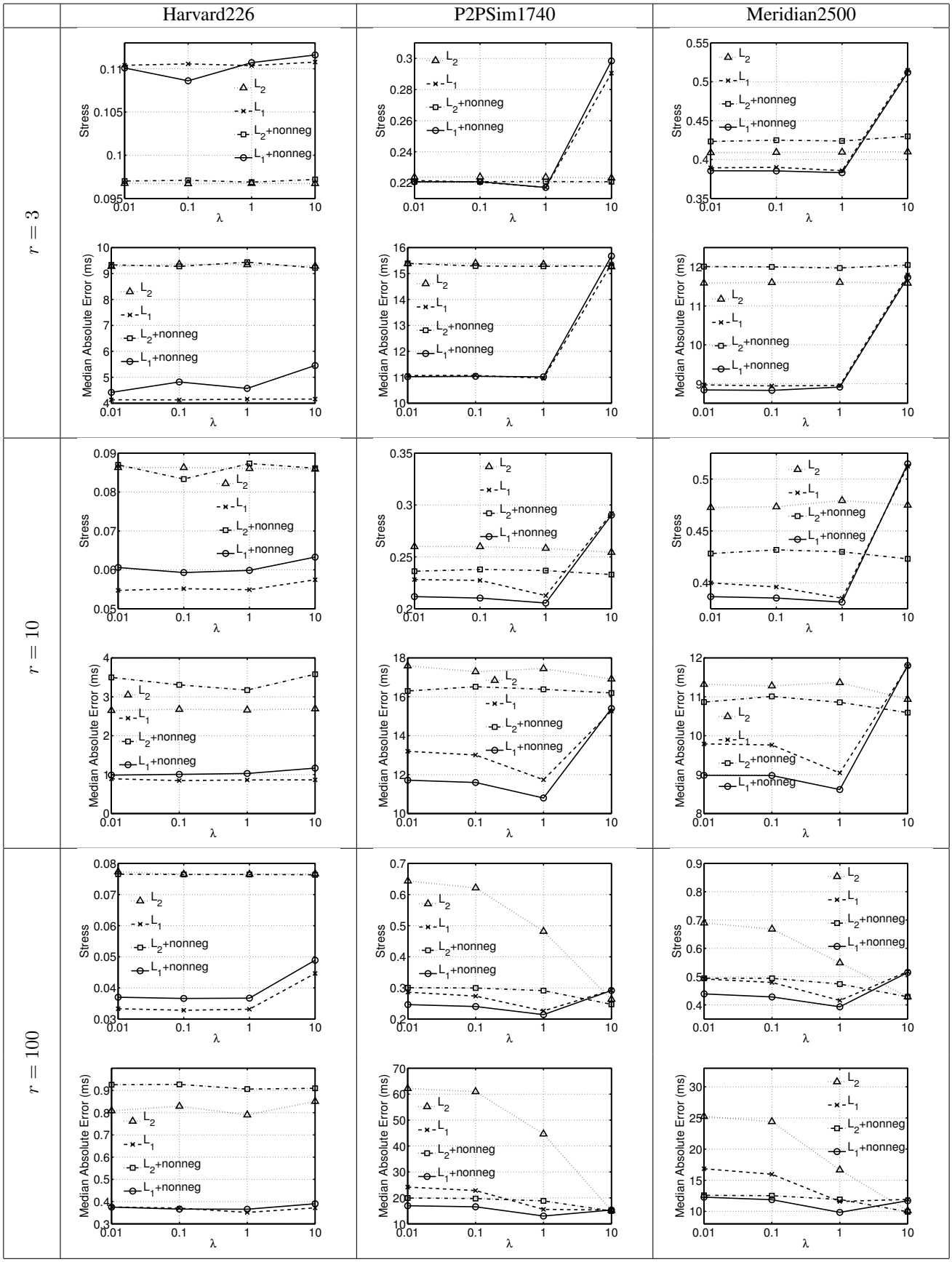


Fig. 6. Impact of parameters. η is adapted by the line search.

the line search to adapt η dynamically. Results are shown in Figure 7. 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[3]. 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 Harvard226, we treat it as $k = 226$.

6) 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 [17]–[19], 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. We experimented with different ks on P2PSim1740 and Meridian2500. For each k, we ran DMFSGD for 10 times, with different random neighbor selections and with different random coordinate initializations, and calculated the mean and the standard deviation of the stress of the 10 runs, shown in Table III. 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 our choice of $k = 32$ for n as large as 2500 is indeed a good tradeoff 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 for Harvard226, we did the 10 runs with only different random coordinate initializations, with no neighbor selection.

TABLE III
MEAN AND STANDARD DEVIATION OF STRESS

	stress	$k = 8$	$k = 16$	$k = 32$	$k = 64$
P2PSim1740	mean	0.341	0.240	0.207	0.187
	std	0.006	0.002	0.001	0.001
Meridian2500	mean	0.505	0.430	0.380	0.359
	std	0.007	0.004	0.002	0.002
Harvard226	mean	0.060			
	std	0.001			

[3]For P2PSim1740 and Meridian2500, 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 Harvard226, this number is significantly different for different nodes because the paths were passively probed with uneven frequencies.

D. Comparisons with Vivaldi

Among numerous approaches to network distance prediction, we consider Vivaldi [7] as the state of the art because of its accuracy and its practicability. To the best of our knowledge, Vivaldi is the only system that has been actually adopted in a real application, Azureus [27]. Other approaches such as GNP [6] and IDES [8] 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 Harvard226 dataset). As mentioned in Section VI, we consider these landmark-based systems as a special variation of a generic decentralized model.

In this paper, we only compare our DMFSGD algorithm with Vivaldi. To address the measurement dynamics and the skewed neighbor updates, we adopted the Vivaldi implementation in [11] [4] when dealing with the Harvard226 dataset. The conventional Vivaldi in [7] was adopted to deal with the other two datasets. We refer to the former as Harvard Vivaldi to make the distinction. In addition, despite the impracticality, we also demonstrate the flexibility of the DMFSGD algorithm in dealing with the landmark-based architecture, referred to as DMFSGD Landmark, by forcing each node to only select the landmarks as neighbors. Note that we only ran DMFSGD Landmark on P2PSim1740 and Meridian2500. To make the comparison fair, 32 landmarks were randomly selected.

Figure 8 shows the comparisons between DMFSGD, Vivaldi/Harvard Vivaldi and DMFSGD Landmark. It can be seen that under $k = 32$, DMFSGD either outperforms or is competitive with Vivaldi on different criteria. On Harvard226, 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 P2PSim1740 and Meridian2500, while the stress of DMFSGD and Vivaldi is similar, moderate improvements by DMFSGD were achieved on either the MAE or the cumulative distributions of REE. Note that DMFSGD and DMFSGD Landmark perform similarly.

The superiority of DMFSGD on Harvard226 is interesting. On the one hand, it demonstrates the usability of DMFSGD as Harvard226 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, we performed another experiment on P2PSim1740 and Meridian2500 with $k = 128$, shown in Figure 9. It can be seen that more decent improvements of DMFSGD over Vivaldi are achieved. 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 8 and in Figure 9 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

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

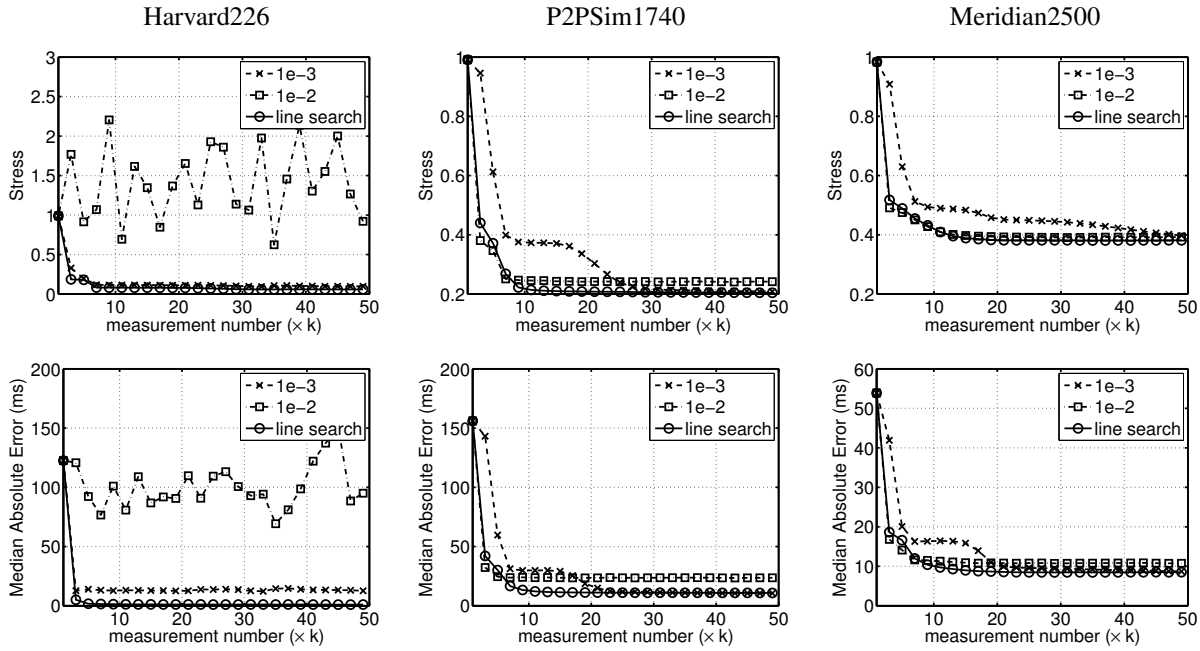


Fig. 7. Impact of η. k is treated as 226 for Harvard226 and $k = 32$ for P2PSim1740 and Meridian2500.

measurements are probed passively as in the Harvard226 dataset. Note also that in Figure 8, while DMFSGD converges fast, Vivaldi appears to converge slightly faster.

E. Drift of DMFSGD Coordinates

In [11], 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. While the translation of the coordinates has been later overcome by adding a gravity term [11], the rotation still remains in Vivaldi.

We expect a similar behavior by DMFSGD due to a similar invariance property described in eq. 5. 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.,

$$\hat{D} = XY^T = (XR)(YR)^T, \tag{29}$$

where R is any arbitrary orthogonal matrix with $RR^T = I$. Thus, the pair of XR and YR are equivalent to the pair of X and Y in the sense that they not only produce the same $\hat{D}$ 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 [41] that stabilizes the Vivaldi coordinates by progressively eliminating the prediction 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.

VIII. Conclusions and Future Works

This paper presents a novel approach to network distance prediction by low-rank matrix factorization. The success of the approach roots both in the exploitation of the correlations across distance measurements between network nodes and in the stochastic optimization which enables a fully decentralized architecture. A so-called Decentralized Matrix Factorization by Stochastic Gradient Descent (DMFSGD) algorithm is proposed to solve the distance prediction problem. The algorithm is simple, with no infrastructure, scalable, able to deal with dynamic measurements in large-scale networks, and accurate, often superior to and at least competitive with Vivaldi. Extensive experiments on various RTT datasets, particularly on one with real data from Azureus, demonstrate the potential of the algorithm being utilized by Internet applications.

Our DMFSGD approach is flexible and can easily be extended to other network metrics such as available bandwidth [42]. An interesting topic is to study which metrics are suitable for our matrix completion framework.

Acknowledgments

The authors thank the anonymous reviewers for their helpful feedback which improved the paper. The authors thank Laurent Mathy for proof-reading the paper. This work was partially supported by the EU under project FP7-Fire ECODE, by the European Network of Excellence PASCAL2 and by the Belgian network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office. The scientific responsibility rests with its authors.

References

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

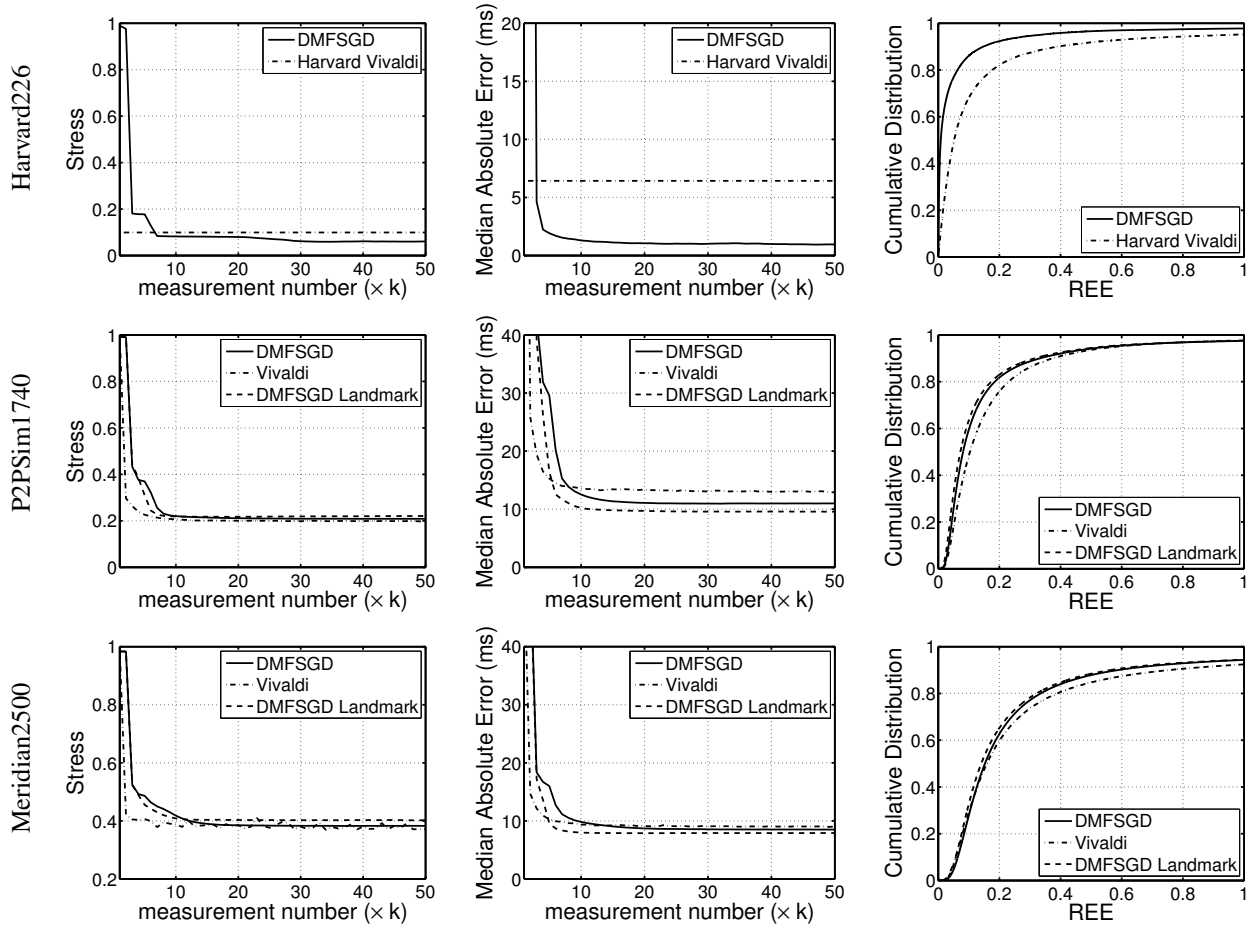
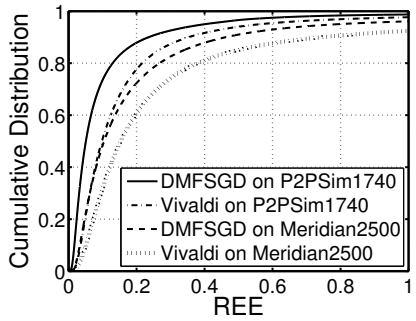


Fig. 8. Comparison of DMFSGD and Vivaldi under $k = 32$. Note that as the implementation of Harvard Vivaldi only outputs the results in the end of the simulation, the final stress and the final MAE are plotted as a constant.

	$k = 128$	P2PSim1740	Meridian2500
stress	Vivaldi	0.186	0.361
	DMFSGD	0.176	0.347
MAE (ms)	Vivaldi	12.9	8.8
	DMFSGD	7.0	5.8

(a) Stress and MAE



(b) Cumulative Distribution of REE

Fig. 9. Comparison of DMFSGD and Vivaldi under $k = 128$.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. of ACM SIGCOMM*, San Diego, CA, USA, Aug. 2001, pp. 161–172.

[3] F. Dabek, "A distributed hash table," Ph.D. dissertation, Massachusetts Institute of Technology, Nov. 2005.

[4] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. of IEEE INFOCOM*, New York, NY, USA, Jun. 2002, pp. 1190–1199.

[5] M. J. Freedman, K. Laskhminarayanan, and D. Mazières, "OASIS: Anycast for any service," in *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, San Jose, CA, May 2006, pp. 10–23.

[6] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. of IEEE INFOCOM*, New York, NY, USA, Jun. 2002, pp. 170–179.

[7] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. of ACM SIGCOMM*, Portland, OR, USA, Aug. 2004, pp. 15–26.

[8] Y. Mao, L. Saul, and J. M. Smith, "IDES: An Internet distance estimation service for large networks," *IEEE Journal On Selected Areas in Communications*, vol. 24, no. 12, pp. 2273–2284, Dec. 2006.

[9] 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, pp. 15–26.

[10] B. Donnet, B. Gueye, and M. A. Kaafar, "A survey on network coordinates systems, design, and security," *IEEE Communication Surveys and Tutorial*, vol. 12, no. 4, pp. 488–503, 2010.

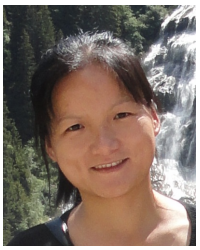
[11] J. Ledlie, P. Gardner, and M. I. Seltzer, "Network coordinates in the wild," in *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, Cambridge, Apr. 2007, pp. 22–35.

[12] 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, Apr. 2005, pp. 236–250.

[13] S. Lee, Z. Zhang, S. Sahu, and D. Saha, "On suitability of Euclidean embedding of Internet hosts," *SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 1, pp. 157–168, 2006.

[14] G. Wang, B. Zhang, and T. S. E. Ng, "Towards network triangle inequality violation aware distributed systems," in *Proc. of ACM/SIGCOMM*

Internet Measurement Conference, San Diego, CA, USA, Oct. 2007, pp. 175–188.
[15] S. Banerjee, T. G. Griffin, and M. Pias, "The interdomain connectivity of PlanetLab nodes," in *Proc. of the Passive and Active Measurement*, Antibes Juan-les-Pins, France, Apr. 2004, pp. 73–82.
[16] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft, "On the accuracy of embeddings for Internet coordinate systems," in *Proc. of ACM/SIGCOMM Internet Measurement Conference*, Berkeley, CA, USA, 2005, pp. 1–14.
[17] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational Mathematics*, vol. 9, no. 6, pp. 717–772, 2009.
[18] E. J. Candès and Y. Plan, "Matrix completion with noise," *Proc. of the IEEE*, vol. 98, no. 6, pp. 925–936, 2010.
[19] R. H. Keshavan, S. Oh, and A. Montanari, "Matrix completion from a few entries," *CoRR*, vol. abs/0901.3150, 2009.
[20] L. Tang and M. Crovella, "Virtual landmarks for the Internet," in *Proc. of ACM/SIGCOMM Internet Measurement Conference*, Miami, FL, USA, Oct. 2003, pp. 143–152.
[21] D. B. Chua, E. D. Kolaczyk, and M. Crovella, "Network kriging," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2263–2272, Dec. 2006.
[22] Y. Chen, D. Bindel, H. Song, and R. H. Katz, "An algebraic approach to practical and scalable overlay network monitoring," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 55–66, Aug. 2004.
[23] N. S. Nati and T. Jaakkola, "Weighted low-rank approximations," in *International Conference on Machine Learning*, Washington, DC USA, 2003, pp. 720–727.
[24] A. M. Buchanan and A. W. Fitzgibbon, "Damped newton algorithms for matrix factorization with missing data," in *Computer Vision and Pattern Recognition*, vol. 2, San Diego, CA, USA, 2005, pp. 316–322.
[25] S. Shalev-Shwartz, A. Gonen, and O. Shamir, "Large-Scale Convex Minimization with a Low-Rank Constraint," in *International Conference on Machine Learning*, Bellevue, Washington, USA, 2011, pp. 329–336.
[26] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*, D. Saad, Ed. Cambridge University Press, 1998.
[27] Vuze Bittorrent, http://www.vuze.com/.
[28] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao, "A measurement study of Internet delay asymmetry," in *Proc. of the Passive and Active Measurement*, Cleveland, OH, USA, Apr. 2008, pp. 182–191.
[29] Y. He, M. Faloutsos, S. Krishnamurthy, B. Huffaker, Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker, "On routing asymmetry in the Internet," in *Proc. of IEEE Globecom*, vol. 2, St. Louis, Missouri, USA, 2005, pp. 904–909.
[30] B. Wong, A. Slivkins, and E. Sirer, "Meridian: A lightweight network location service without virtual coordinates," in *Proc. of ACM SIGCOMM*, Philadelphia, Pennsylvania, USA, Aug. 2005, pp. 85–96.
[31] G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed. Johns Hopkins University Press, 1996.
[32] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2001, pp. 556–562.
[33] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Scalable collaborative filtering approaches for large recommender systems," *Journal of Machine Learning Research*, vol. 10, pp. 623–656, Jun. 2009.
[34] D. Bertsekas, *Nonlinear programming*. Athena Scientific, 1999.
[35] C. Hennig and M. Kutlukaya, "Some thoughts about the design of loss functions," *REVSTAT–Statistical Journal*, vol. 5, no. 1, pp. 19–39, 2007.
[36] Q. Ke and T. Kanade, "Robust L_1 norm factorization in the presence of outliers and missing data by alternative convex programming," in *Computer Vision and Pattern Recognition*, vol. 2, San Diego, CA, USA, 2005, pp. 592–599.
[37] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural Computation*, vol. 19, no. 10, pp. 2756–2779, Oct 2007.
[38] 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, Nov. 2002, pp. 5–18.
[39] I. Borg and P. Groenen, *Modern multidimensional scaling : theory and applications*. Springer, 2005.
[40] *matlab mdscale*, http://www.mathworks.com/help/stats/mdscale.html.
[41] G. Wang and T. S. E. Ng, "Distributed algorithms for stable and secure network coordinates," in *Proc. of ACM/SIGCOMM Internet Measurement Conference*, Vouliagmeni, Greece, Oct. 2008, pp. 131–144.
[42] Y. Liao, W. Du, P. Geurts, and G. Leduc, "Decentralized prediction of end-to-end network performance classes," in *Proc. of CoNEXT*, Tokyo, Japan, Dec. 2011, pp. 14:1–14:12.



Yongjun Liao is a PhD student at Department of Electrical Engineering and Computer Science, University of Liège, Belgium. She received her B.S. in 1999 and M.S. in 2002, both from Department of Computer Science, Guangxi University, China. Before joining the networking group RUN in University of Liège in 2007, she had worked as a software engineer in a small computer company in Beijing, China. Her main research interests are the applications of machine learning techniques to computer networking problems, specifically the prediction of end-to-end network performance in large-scale networks.

Wei Du is a visiting researcher at the group Research Unit in Networking (RUN) in University of Liège, Belgium. He received his B.S. in 1997 from Tianjin University, China, and PhD in 2002 from Institute of Computing Technology, Chinese Academy of Sciences, China. Since graduation, he has been working as postdoctoral researcher at INRIA, France, Hamburg University, Germany, University of Liège, Belgium, and University of Innsbruck, Austria. His main research interests are computer vision and machine learning.

Pierre Geurts is an assistant professor in the EECS department of the University of Liège, Belgium. He graduated as an electrical (computer science) engineer in 1998 and received the PhD degree in applied sciences in 2002. From 2006 to 2011, he was research associate of the FNRS (Belgium). His research interests concern the design of, computationally and statistically efficient, supervised and semi-supervised learning algorithms in order to exploit structured input and output spaces (sequences, images, time-series, graphs), with applications in bioinformatics, computer vision, and computer networks.

Guy Leduc is a full professor in the EECS department of the University of Liège, Belgium, and is since 1997 the head of the Research Unit in Networking (RUN). He graduated as an electrical (electronics) engineer in 1983 and got his PhD in computer science in 1991.

His research field is computer networks, and his main research interests are Network Coordinate Systems, overlays, traffic engineering, resilience, multimedia, congestion control, and autonomic/active/programmable networks. His research unit is or has been involved in European projects such as ECODE on cognitive networking, ResumeNet on Resilient Networking, ANA on autonomic networking, TOTEM on an open-source toolbox for traffic engineering, and the E-NEXT European network of excellence.

Since 2007 he has been the chairman of the IFIP Technical Committee (TC6) on Communications Systems. He is an area editor of the Elsevier Computer Communications journal, and a steering committee member of the IFIP Networking Conference.