Towards a Robust Framework of Network Coordinate Systems

Linpeng Tang[1,2], Zhiyong Shen[2,⋆], Qunyang Lin[2], and Junqing Xie[2]

[1] Shanghai Jiao Tong University
chnttlp@gmail.com
[2] HP Labs, China
{peng.tang,zhiyongs,qun-yang.lin,jun-qing.xie}@hp.com

Abstract. Network Coordinate System (NCS) is an efficient and scalable mechanism to predict latency between any two network hosts based on historical measurements. Most NCS models, such as metric space embedding based, like Vivaldi, and matrix factorization based, like DMF and Phoenix, use squared error measure in training which suffers from the erroneous records, i.e. the records with large noise. To overcome this drawback, we introduce an elegant error measure, the Huber norm to network latency prediction. The Huber norm shows its robustness to the large data noise while remaining efficiency of optimization. Based on that, we upgrade the traditional NCS models into more robust versions, namely Robust Vivaldi model and Robust Matrix Factorization model. We conduct extensive experiments to compare the proposed models with traditional ones and the results show that our approaches significantly increase the accuracy of network latency prediction.

Keywords: Network Coordinate Systems, Robust Error Measure, Metric Space Embedding, Matrix Factorization.

1 Introduction

The prediction of network latency, e.g. Round-Trip Time(RTT), has been a hot research topic over the last few years. Predicting network latencies between two network hosts without involving physical measurements may benefit many networking applications, like geometric routing[14], large scale online game systems [4], locality-aware data center selection[6], and P2P file sharing.

One way to predict network latency is via a Network Coordinate System (NCS)[9,17,15,8]. NCS tries to assign each network host some coordinates representing its virtual location in the network and the latency between two hosts is computed using a *prediction function* over the coordinates. To learn the coordinates for each host, we only need to measure a small fraction of all the pairs of the hosts and adjust the coordinates in order to minimize the *prediction error* on these records. Since the coordinates of one host are just a constant number of

⋆ Corresponding author. 5F, Block A, SP Tower, No. 1, ZhongGuanCun East Road, Haidian District Beijing 100084, P.R. China values, from Valiant's learning theory [18] we could learn the coordinates with just $O(N)$ measurements and predict the N^2 latencies between all host pairs, which makes the system scalable.

NCS approaches could be categorized into two classes: metric space embedding models and matrix factorization models. One of the most representative metric space embedding based approach is Vivaldi [9], which maps the network hosts into a Euclidean space and approximates network latencies with the Euclidean distances between the image points. In matrix factorization models, such as Phoenix [8] and DMF [15], the pairwise latency matrix is approximated by the factorization of two low-rank matrices, one for the source hosts and the other for the destination hosts. Most of these traditional NCS's use the ℓ^2-norm[1] as the measure of the prediction error, due to its simplicity in model learning. However, the complexity of the Internet may cause extremely large noise in the measurement of the latency between hosts, and the quadratic ℓ^2-norm shows high sensitivity to such erroneous data records. In this paper, we address the issue of large data noise based on an elaborately designed error measure—the Huber norm, which is more robust for the erroneous data records while remaining simplicity of learning the model. Based on the Huber norm, we propose a robust framework of NCS and further increase the accuracy of network latency prediction. The key contributions of this paper are threefold:

1. To the best of our knowledge, we are the first to introduce and study the Huber norm for network latency prediction, based on which we upgrade the traditional Vivaldi and matrix factorization models to more robust versions.
2. We derive two algorithms to learn the robust models—stochastic gradient descent and alternative damped-Newton method. The former has simple implementation and runs faster in terms of CPU time, while the latter takes fewer iterations for convergence.
3. We conduct extensive experiments to demonstrate that our robust models significantly outperform the traditional ones. Specifically, we find the robust version of Vivaldi shows much higher accuracy than other methods when neighbors of one host are fewer than 32 and may be the preferred model in many applications.

1.1 Preliminaries

Before we introduce the NCS approaches, let's formally define the problem of network coordinating. Assume that we have N network hosts. We may use an $N \times N$ matrix Y to denote the pairwise latency matrix, and let W denote the indices set of known entries, i.e., $Y_{i,j}$ is known(measured) for $(i,j) \in W$ and generally $|W| \ll N \times N$. For each entry in the latency matrix Y, whether or not known, we may predict its value using $\hat{Y}_{i,j} = pred_{\mathcal{C}}(i,j)$, where $pred_{\mathcal{C}}(i,j)$ is a universal representation of the prediction function and the prediction is made based on the virtual coordinates assigned to the hosts, noted as $\mathcal{C}$. The problem is now how to assign and adjust the coordinates for each network host in order to make

[1] The ℓ^2-norm of an error vector $\mathbf{e} = (e_1, e_2, ..., e_n)$ is defined as $\|e\|_2 = \sqrt{\sum_{i=1}^{n}(e_i)^2}$.

the predicted latency matrix $\hat{Y}$ as *close* to the matrix of true measurements Y as possible. We use the prediction error measure function $\phi(\hat{Y}, Y)$ to characterize the closeness between the $\hat{Y}$ and Y. Then in a NCS, we generally aim to solve the following optimization problem

$$\mathcal{C}^* = \underset{\mathcal{C}}{\operatorname{argmin}}\, \phi(\hat{Y}, Y) \tag{1}$$

Both the prediction function $pred_{\mathcal{C}}(i, j)$ and the error measure function $\phi(\hat{Y}, Y)$ should be specified for a NCS.

2 Related Work

2.1 Metric Space Embedding Models

In metric space embedding models, each host is associated with a position in a metric space, such as Euclidean space [9] and hyperbolic space [16], etc. The latency between two hosts is then estimated as the distance between their positions in the space. Vivaldi [9] is the most representative metric space embedding based approach, whose variants have been employed in real-world systems, such as Htrae [4]. In this model, the network hosts are embedded into a D-dimensional Euclidean space. Assume that two nodes i, j have embedded coordinates $X_i = (X_{i,1}, ..., X_{i,D})$ and $X_j = (X_{j,1}, ..., X_{j,D})$. We may calculate the Euclidean distance between i, j as $\|X_i - X_j\|_2$.

In Vivaldi, the authors also define quantities of *heights* $H = (h_1, h_2, ..., h_N)$ $(h_i \geq 0)$ for each host to characterize the host's intrinsic contribution to the latency, say the latency from a host to the nearest ISP. The prediction function is then defined as

$$pred_{X,H}(i, j) := \|X_i - X_j\|_2 + h_i + h_j, \tag{2}$$

so we have $\hat{Y}_{i,j} = \|X_i - X_j\|_2 + h_i + h_j$. The coordinates are learned by optimization algorithms so as to minimize the summation of prediction errors over the known entries in Y. Formally, in Vivaldi, we achieve the coordinates X and H by solving the the following optimization problem with an ℓ^2-norm objective function

$$X^*, H^* = \underset{X,H}{\operatorname{argmin}}\, \phi(\hat{Y}, Y) = \underset{X,H}{\operatorname{argmin}} \left\| \hat{Y} - Y \right\|_2 \tag{3}$$

$$= \underset{X,H}{\operatorname{argmin}} \sum_{(i,j)\in W} [\|X_i - X_j\|_2 + h_i + h_j - Y_{i,j}]^2 \tag{4}$$

The above problem could be solved using *stochastic gradient descent* [5] and due to the space limitation, we refer to [9] for details.

2.2 Matrix Factorization Models

The representative matrix factorization NCS approaches include IDES [17], Phoenix [8] and DMF [15] etc. In this kind of models, we generally seek the following approximation for the latency matrix Y.

$$Y \approx UV^T$$

The two factor matrices U, V represent the out-coordinates and the in-coordinates of the hosts. Specifically, the i-th row of U is the out-coordinate of host i and the jth row of V is the in-coordinate of node j. Let $\hat{Y} = UV^T$, so $\hat{Y}_{i,j} = \sum_{k=1}^{K} U_{i,k}V_{j,k}$. The prediction function here is specified as

$$pred_{U,V}(i,j) := \sum_{k=1}^{K} U_{i,k}V_{j,k} = (UV^T)_{i,j}. \tag{5}$$

Similar to Vivaldi, we may find U and V by minimizing the prediction error $\phi(\hat{Y}, Y)$ over the known entries, and formally we have

$$U^*, V^* = \underset{U,V}{\operatorname{argmin}}\, \phi(\hat{Y}, Y) = \underset{U,V}{\operatorname{argmin}} \left\| \hat{Y} - Y \right\|_2 \tag{6}$$

$$= \underset{U,V}{\operatorname{argmin}} \sum_{(i,j)\in W} [(UV^T)_{i,j} - Y_{i,j}]^2 \tag{7}$$

From (6) and (3), we may see that traditional NCS's mostly use l^2-norm in their objective function. This choice of prediction error measure has its easiness for optimization. In the next subsection, we'll discuss whether this is a proper choice for network latency prediction.

2.3 Issue of Data Noise

We must also, however, recognize the inherent complexity of network latencies. Because of the existence of inefficient routing rule, network congestions, malicious attack etc., many of data records we get and use to build the NCS may contain extremely large noise. If the NCS tries to fit these data records, it may deviate from the true distance model behind the whole network. Many people have recognized this phenomenon, and try to overcome it by introducing a "faith" factor on the host or the records. For example, Vivaldi introduced a weight in $[0, 1]$ on each host to indicate how reliable its coordinates are. Similarly, Phoenix introduced a weight on each record indicating how reliable it is. Hosts or records with smaller weights will have less impact on the NCS. While such weight based approach can improve the stability and accuracy of NCS, it is very heuristic and often involves several parameters in the model that make tuning difficult.

We attack the issue of data noise from another direction by leveraging a noise robust error measure—the Huber norm, based on which we propose the robust versions of Vivaldi and matrix factorization models as described in the next section.

3 Algorithm Design

3.1 Huber Norm

A simple choice of robust measure is the ℓ^1-norm[2], which has shown considerably less sensitivity to large measurement errors than ℓ^2-norm measures[13]. However, ℓ^1-norm is not continuously differentiable so its numerical minimization is difficult. In [12], Huber presents an elegant error-measure defined as (8) and illustrated in Figure 1.

$$M_\epsilon(r) = \begin{cases} \frac{r^2}{2\epsilon}, & |r| \le \epsilon \\ |r| - \frac{\epsilon}{2}, & |r| \ge \epsilon \end{cases} \tag{8}$$

The Huber norm is a combination of ℓ^1/ℓ^2 norm (see in Figure 1). The tradeoff of ℓ^1 or ℓ^2 is controlled by the parameter ϵ : for small errors $|r| \le \epsilon$, it assumes the ℓ^2 norm, while for large errors $|r| > \epsilon$, it assumes the ℓ^1 norm. We'll empirically investigate the role of parameter ϵ in our models in Section 4.3. Compared to ℓ^2 norm, the Huber norm is more robust since large errors only has a linear impact. On the other hand, it is easier to optimize than ℓ^1 norm in some sense because it is continuously differentiable.

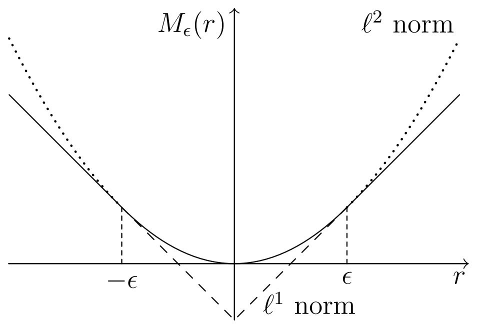


Fig. 1. The Huber norm

By leveraging the Huber norm, the learning objective of NCS (1) becomes

$$\mathcal{C}^* = \underset{\mathcal{C}}{\operatorname{argmin}} \sum_{(i,j)\in W} M_\epsilon(pred_{\mathcal{C}}(i,j) - Y_{i,j}) \tag{9}$$

In the next two subsections we will describe how we specify and solve (9) for the robust versions of Vivaldi and matrix factorization models, respectively.

3.2 Robust Vivaldi

Substituting (2) into (9), we may train the model to learn the coordinates in the Euclidean space X and the heights H via optimizing

$$X^*, H^* = \underset{X,H}{\operatorname{argmin}} \sum_{(i,j)\in W} M_\epsilon[\|X_i - X_j\|_2 + h_i + h_j - Y_{i,j}] \tag{10}$$

[2] The ℓ^1-norm of an error vector $\mathbf{e} = (e_1, e_2, ..., e_n)$ is defined as $\|e\|_1 = \sum_{i=1}^n |e_i|$.

We leverage two alternative methods, stochastic gradient descent and alternative damped-Newton to solve the above problem.

Stochastic Gradient Descent: One way to solve the problem is by stochastic gradient descent [5]. For each measurement record $Y_{i,j}$, we do the following update on the coordinates X_i and heights h_i:

$$F_{i,j} = M_\epsilon(\|X_i - X_j\| + h_i + h_j - Y_{i,j}) \tag{11}$$

$$X_{i,d} \leftarrow X_{i,d} - \delta \frac{\partial}{\partial X_{i,d}} F_{i,j} \qquad (d = 1 \cdots D) \tag{12}$$

$$h_i \leftarrow \max(0, h_i - \delta \frac{\partial}{\partial h_i} F_{i,j}) \tag{13}$$

where δ is the learning rate. In each round, we first randomly shuffle the training data, and then do the updates as (11), (12), (13).

Alternative Damped-Newton: We then utilize another optimization scheme– alternative damped-Newton method [7].

Instead of considering one sample at a time, alternative damped-Newton method considers all neighbors[3] of one host at the same time. Specifically, for one host i, fixing the coordinates of other hosts, it tries to optimize the sum of prediction errors from i to all its neighbors:

$$F_i = \sum_{j:(i,j)\in W} M_\epsilon(\|X_i - X_j\|_2 + h_i + h_j - Y_{i,j}) \tag{14}$$

To achieve this efficiently, we first consider Newton's method. Let $\mathcal{G} = \nabla_{X_i} F_i$ be the gradient, $\mathcal{H} = \mathcal{H}_{X_i} F_i$ be the Hessian. Then Newton's Method updates $X_i \leftarrow X_i - \mathcal{G} \backslash \mathcal{H}$[4]. However, since $M''_\epsilon(x) = 0$ for $|x| < \epsilon$, $\mathcal{H}$ can often be singular and $\mathcal{G} \backslash \mathcal{H}$ undefined. To overcome this problem, we use Damped-Newton Method. Specifically, we introduce the *damping factor* λ and modify the update rule to:

$$X_i \leftarrow X_i - \mathcal{G} \backslash (\mathcal{H} + \lambda I) \tag{15}$$

Note that when λ is small, the update rule approximates a Newton update, but when λ grows larger, the update becomes similar to a gradient descent with learning rate $1/\lambda$. We use an adaptive scheme to choose a proper λ—starting with $\lambda = 1$, try $\lambda \leftarrow 2\lambda$ until F_i decreases or λ becomes too large, and this finishes one update.

3.3 Robust Matrix Factorization

By instituting (5) into (9), we get the objective of our Robust Matrix Factorization model.

$$U^*, V^* = \underset{U,V}{\operatorname{argmin}} \sum_{(i,j)\in W} M_\epsilon[(UV^T)_{i,j} - Y_{i,j}] \tag{16}$$

[3] The neighbors of one host i are the hosts j s.t. $Y_{i,j}$ is known, noted as $j : (i,j) \in W$.
[4] $X = A \backslash B$ is the solution of $A \times X = B$.

Both stochastic gradient descent and alternative damped-Newton method can be applied in similar ways. Note that we only give the update rules for the out-coordinates U and the rules for V are symmetric.

Stochastic Gradient Descent: Let $F_{i,j}$ denote the prediction error of record $Y_{i,j}$ (17). We may minimize it using gradient descent on U_i.

$$F_{i,j} = M_\epsilon(\sum_{d=1}^{D} U_{i,d}V_{j,d} - Y_{i,j}) \tag{17}$$

$$U_{i,d} \leftarrow U_{i,d} - \delta \frac{\partial}{\partial U_{i,d}} F_{i,j} \qquad (d = 1 \cdots D) \tag{18}$$

Alternative Damped-Newton: Let F_i denote the sum of prediction error from i to all its neighbors (19). We try to minimize F_i by updating U_i with alternative damped-Newton method (15). Again, assuming $\mathcal{G} = \nabla_{U_i} F_i$ and $\mathcal{H} = \mathcal{H}_{U_i} F_i$, we have

$$F_i = \sum_{j:(i,j)\in W} M_\epsilon(\sum_{d=1}^{D} U_{i,d}V_{j,d} - Y_{ij}) \tag{19}$$

$$U_i \leftarrow U_i - \mathcal{G}\backslash(\mathcal{H} + \lambda I) \tag{20}$$

3.4 Discussion

As can be seen, stochastic gradient descent and alternative damped-Newton actually work in two fashions. The former considers one measurement record at a time while the latter update the coordinate of one host based on the current coordinates of all its neighbors. Intuitively, methods that consider the coordinates of all neighbors of one node converges faster, i.e., requiring fewer iterations to reach a optimal value. This property is especially desirable when each node maintain its own coordinates and communicates with its neighbors to adjust the coordinates. Faster convergence, in this case, means that it takes less time for the dynamic system to become stable.

However, stochastic gradient descent also has its merits. Since it has no need to compute the Hessian, and do the matrix inversion, the computational cost of one iteration is extremely cheap. Therefore, although it may take more iterations, it is still considerably faster than alternative damped-Newton method when running on one computer. In Section 4.4, we'll further investigate the convergence property of these two methods.

4 Experiments

In this section we empirically compare performance of our models: Robust Vivaldi and Robust MF, with traditional models: Vivaldi, DMF, IDES and Phoenix. We use three large public data sets for our experiments: Meridian [2], P2PSim

[3], Harvard [1], which are all collected with King method[11]. The data sets are preprocessed by removing some hosts to guarantee that all the hosts have at least 32 neighbors. In addition, we are predicting the RTT between hosts, which should be symmetric (not considering factors such as network congestion), so for the original latency matrix Y, we transform it to $(Y + Y^T)/2$. After these preprocessing steps, the three data sets have 2,500, 1740 and 1818 hosts and average latency of 91.8ms, 75.8ms, 85.7ms, respectively.

4.1 Overview of the Experiments

We have three parameters to be specified in the experiments: K – the number of measured neighbors of a host, D – the dimension of the coordinates and ϵ – the tradeoff parameter between ℓ^1 and ℓ^2 norm in the Huber norm (8). Note that in all experiments, the K neighbors of any host are chosen randomly. In the experiments, we consider the cases when $K = 16, 32, 64$ and the choice of D are chosen by experiments to let the algorithms to achieve best accuracy. We found that Robust-Vivaldi always works best when $D = 2$. (This coincides the finding of [9] that increasing the dimension can't significantly improve the prediction accuracy in Vivaldi model.) The matrix factorization methods should have a slightly lower dimension when K are small in order to avoid overfitting, but a higher dimension when K are large in order to improve accuracy. So we set $D = 5$ when $K = 16$, $D = 6$ when $K = 32$ and $D = 7$ when $K = 64$. In Robust Vivaldi and Robust-MF, we set $\epsilon = 7$ and in Section 4.3 we will conduct further investigation on the choice of ϵ.

In Section 4.2, we give the results on the distribution of absolute error. We then used other two evaluation criteria for later experiments, Mean Absolute Error (MAE) and Rooted Mean Squared Error (RMSE), defined as follows

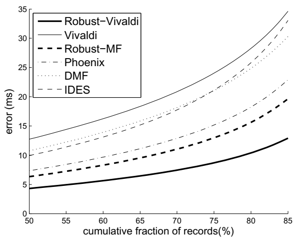
$$\text{MAE} = \underset{(i,j)\in W^*}{\text{average}} \left(\left| \hat{Y}_{i,j} - Y_{i,j} \right| \right) \qquad \text{RMSE} = \sqrt{\underset{(i,j)\in W^*}{\text{average}} \left[(\hat{Y}_{i,j} - Y_{i,j})^2 \right]} \tag{21}$$

where, W^* denotes the indices of a held out test set. For both of the two criteria, smaller values mean better performance.

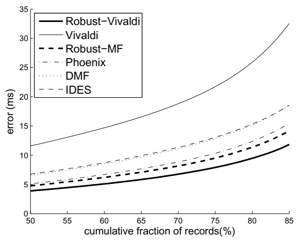
4.2 Comparison on Prediction Accuracy

In this subsection, we compare the prediction accuracy of our algorithms with Phoenix, DMF, IDES and Vivaldi. The comparison results of error distributions are given in Figure 2(a)-(i) and we also specify the comparison of the 80-percentile error in Figure 2(j). From these results, we see that robust methods significantly outperform their non-robust counterparts. This observation verifies the robustness of the Huber norm in the application of network latency prediction.

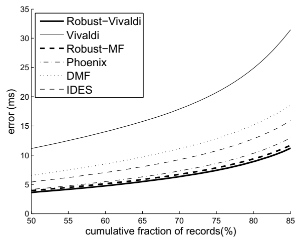
It is also remarkable that although Vivaldi does poorly in all our test cases (which coincides with the conclusion of Phoenix [8] and DMF [15] that matrix factorization supersedes (non-robust) Vivaldi), Robust Vivaldi achieves much



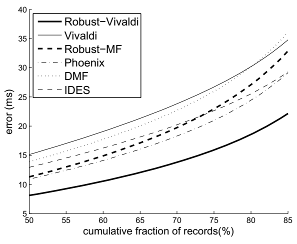
(a) P2PSim, K=16



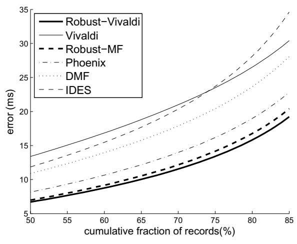
(b) P2PSim, K=32



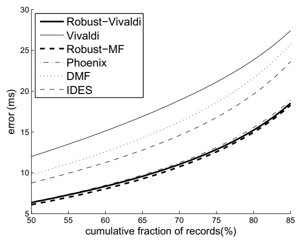
(c) P2PSim, K=64



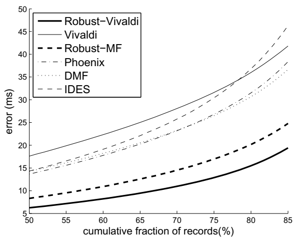
(d) Meridian, K=16



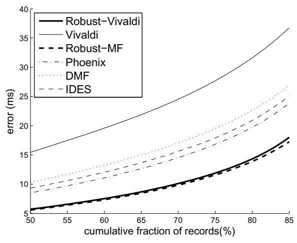
(e) Meridian, K=32



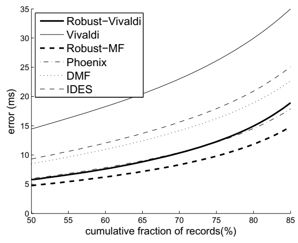
(f) Meridian, K=64



(g) Harvard, K=16



(h) Harvard, K=32



(i) Harvard, K=64

	$K = 16$			$K = 32$			$K = 64$		
	R-Vivaldi	R-MF	Others	R-Vivaldi	R-MF	Others	R-Vivaldi	R-MF	Others
P2PSim	**10.38**	15.62	18.24	**9.46**	11.35	12.35	**8.91**	9.36	10.31
Meridian	**18.57**	27.10	24.43	**15.82**	16.76	18.97	15.20	**14.95**	15.52
Harvard	**15.49**	20.18	30.67	14.32	**13.89**	19.74	14.94	**11.83**	14.53

(j) 80-percentile error (ms)

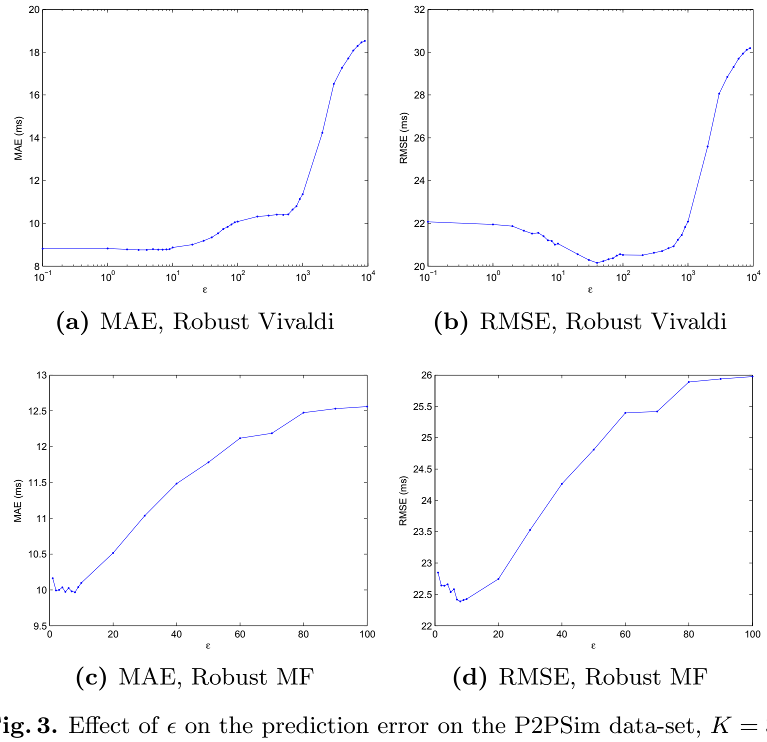
Fig. 2. Results of error distribution shown in (a)-(i). The x-axis is the cumulative fraction of records (in percentiles), and the y-axis is the corresponding prediction error (in milliseconds). In Table (j), we compare the 80-percentile error, where *R-Vivaldi* stands for Robust Vivaldi, *R-MF* stands for Robust MF and *Others* stands for the best result of IDES, Phoenix, DMF and Vivaldi in each test case.

higher accuracy than other methods when K is small. When $K = 16$, it supersedes all other methods by 25%–30% in terms of 80-percentile error in all 3 data sets (see in Figure 2(j)). When K is smaller, the advantage is even more obvious. So Robust Vivaldi may be the preferred model in many applications.

The advantage of Robust Vivaldi may profit from that the number of parameters utilized by Vivaldi like models (three for each host) is much smaller than that of matrix factorization based models ($2D$ for each host). According to the principle of Occam's razor, when there is no sufficient training data, a model with fewer parameters faces lower risk of the issue of *overfitting* and may perform better. On the other hand, performance of the methods based on matrix factorization improves gradually as K increases, taking lead when $K = 64$. When we have sufficient training data, matrix factorization may capture more information with more parameters and achieve higher prediction accuracy. In summary, Vivaldi like model is preferred when we have fewer than 32 neighbors for each host, but when we really know much information about the system (say, an online P2P game system), matrix factorization based methods has the potential to provide better accuracy.

4.3 The Choice of ϵ

In Figure 3, we vary ϵ, the threshold between ℓ^1 norm and ℓ^2 norm, and give the results of both MAE and RMSE on the P2PSim data set. According to the



(a) MAE, Robust Vivaldi **(b)** RMSE, Robust Vivaldi

(c) MAE, Robust MF **(d)** RMSE, Robust MF

Fig. 3. Effect of ϵ on the prediction error on the P2PSim data-set, $K = 32$

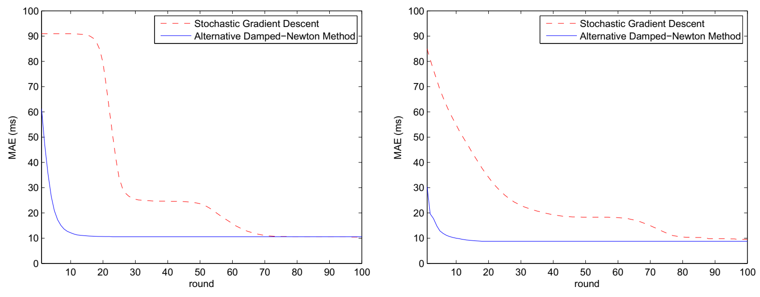
definition in (21), MAE and RMSE actually corresponds to the ℓ^1 and ℓ^2 error measures respectively. So one would expect that if we minimize the ℓ^1 error (set $\epsilon = 0$), MAE would be minimized; and if we use the ℓ^2 error (set a large ϵ), RMSE would be minimized.

Interestingly, this is not the case (see in Figure 3). Neither MAE is best optimized at $\epsilon = 0$, or is RMSE optimized when we increasing ϵ. For Robust Vivaldi, RMSE is best optimized when $\epsilon = 40$. For Robust MF, RMSE is best optimized when $\epsilon = 7$. As ϵ grows beyond this threshold, the performance deteriorates greatly. On the other hand, when ϵ is relatively small ($\epsilon < 10$), the MAE remains stable for Robust Vivaldi. For Robust MF, MAE drops a little when ϵ increases as long as $\epsilon < 7$.

Theoretically speaking, ϵ is the threshold between small errors and large errors. As a rule of thumb, we recommend to use the 80-percentile error as ϵ, so a value between 5–15 is recommended (choose a percentile error as ϵ was recommended in [10]).

4.4 Convergence Analysis

In this section we investigate the convergence of the two optimization schemes, stochastic gradient descent and alternative damped-Newton. Figure 4 shows how the MAE criteria of the two algorithms (on p2psim, $K = 32$) evolve as the number of iterations increases on both Robust Vivaldi and Robust MF models. Eventually the two methods converge to almost the same value. Apparently, alternative damped-Newton converges much faster, reaching a stationary point within 20 iterations. Stochastic gradient descent, on the other hand, taking more (about 80) iterations for convergence. However, due to its efficiency in each round, it still runs much faster than alternative damped-Newton. On a typical 2.4GHz machine, Stochastic Gradient Descent takes less than 10 seconds to compute a coordinate for any data sets we have used while Alternative Newton can take about 2 minutes. These empirical results coincide the discussion in Section 3.4.



(a) On Robust MF Model **(b)** On Robust Vivaldi Model

Fig. 4. Convergence investigation

5 Conclusion and Future Works

In this paper, we argue that the ℓ^2-norm widely used in traditional NCS approaches is not robust for the noisy network latency data. We therefore introduce an elegant error measure, the Huber norm to network latency prediction, based on which we upgrade the traditional Vivaldi and matrix factorization NCS approaches into more robust versions. We conduct extensive experiments and verify the robustness of the upgraded models. According to the results, we recommend Robust Vivaldi as the first choice. However, when there are sufficient historical data, Robust MF is also considerable. We also provide two different learning methods for the models: the alternative damped-Newton method which takes fewer iterations to converge, and the stochastic gradient descent method which is slower in convergence but faster in terms of CPU time. As our future work, we hope to find factors that could capture the noise and further boost the accuracy of NCS.

References

1. Harvard data set, `http://www.eecs.harvard.edu/~syrah/nc/king/lats.n8.gz`
2. Meridian data set, `http://www.cs.cornell.edu/People/egs/meridian/data.php`
3. P2psim data set, `http://pdos.csail.mit.edu/p2psim/kingdata/`
4. Agarwal, S., Lorch, J.: Matchmaking for online games and other latency-sensitive p2p systems. ACM SIGCOMM Computer Communication Review 39(4), 315–326 (2009)
5. Barto, A.G., Anandan, P.: Pattern-recognizing stochastic learning automata. IEEE Transactions on Systems, Man, & Cybernetics (1985)
6. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer networks and ISDN systems 30(1-7), 107–117 (1998)
7. Buchanan, A.M., Fitzgibbon, A.W.: Damped newton algorithms for matrix factorization with missing data. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005, vol. 2, pp. 316–322. IEEE (2005)
8. Chen, Y., Wang, X., Song, X., Lua, E., Shi, C., Zhao, X., Deng, B., Li, X.: Phoenix: Towards an Accurate, Practical and Decentralized Network Coordinate System. In: Fratta, L., Schulzrinne, H., Takahashi, Y., Spaniol, O. (eds.) NETWORKING 2009. LNCS, vol. 5550, pp. 313–325. Springer, Heidelberg (2009)
9. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system. In: ACM SIGCOMM Computer Communication Review, vol. 34, pp. 15–26. ACM (2004)
10. Guitton, A., Symes, W.W.: Robust inversion of seismic data using the huber norm. Geophysics 68(4), 1310 (2003)
11. Gummadi, K., Saroiu, S., Gribble, S.: King: Estimating latency between arbitrary internet end hosts. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment, pp. 5–18. ACM (2002)
12. Huber, P.J.: Robust regression: asymptotics, conjectures and monte carlo. The Annals of Statistics 1(5), 799–821 (1973)
13. Ke, Q., Kanade, T.: Robust l1 norm factorization in the presence of outliers and missing data by alternative convex programming. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005, vol. 1, pp. 739–746. IEEE (2005)

14. Ledlie, J., Pietzuch, P., Mitzenmacher, M., Seltzer, M.: Wired geometric routing. In: Proc. of IPTPS, Citeseer (2007)
15. Liao, Y., Geurts, P., Leduc, G.: Network Distance Prediction Based on Decentralized Matrix Factorization. In: Crovella, M., Feeney, L.M., Rubenstein, D., Raghavan, S.V. (eds.) NETWORKING 2010. LNCS, vol. 6091, pp. 15–26. Springer, Heidelberg (2010)
16. Lumezanu, C., Spring, N.: Playing vivaldi in hyperbolic space (2006)
17. Mao, Y., Saul, L.K., Smith, J.M.: Ides: An internet distance estimation service for large networks. IEEE Journal on Selected Areas in Communications 24(12), 2273–2284 (2006)
18. Valiant, L.G.: A theory of the learnable. Communications of the ACM 27(11), 1134–1142 (1984)