QUINT: On Query-Specific Optimal Networks

Liangyue Li
Arizona State University
liangyue@asu.edu

Yuan Yao
Nanjing University
targenardy@gmail.com

Jie Tang
Tsinghua University
jietang@tsinghua.edu.cn

Wei Fan
Baidu Big Data Lab
fanwei03@baidu.com

Hanghang Tong
Arizona State University
hanghang.tong@asu.edu

ABSTRACT

Measuring node proximity on large scale networks is a fundamental building block in many application domains, ranging from computer vision, e-commerce, social networks, software engineering, disaster management to biology and epidemiology. The state of the art (e.g., random walk based methods) typically assumes the input network is *given* a priori, with the known network topology and the associated edge weights. A few recent works aim to further infer the *optimal edge weights* based on the side information.

This paper generalizes the challenge in multiple dimensions, aiming to learn optimal networks for node proximity measures. First (*optimization scope*), our proposed formulation explores a much larger parameter space, so that it is able to simultaneously infer the optimal network topology and the associated edge weights. This is important as a noisy or missing edge could greatly mislead the network node proximity measures. Second (*optimization granularity*), while all the existing works assume one common optimal network, be it given as the input or learned by the algorithms, exists for all queries, our method performs optimization at a much finer granularity, essentially being able to infer an optimal network that is specific to a given query. Third (*optimization efficiency*), we carefully design our algorithms with a linear complexity wrt the neighborhood size of the user preference set. We perform extensive empirical evaluations on a diverse set of 10+ real networks, which show that the proposed algorithms (1) consistently outperform the existing methods on *all* six commonly used metrics; (2) empirically scale *sub-linearly* to billion-scale networks and (3) respond in a fraction of a second.

Keywords

Node proximity; Optimal networks

KDD '16, August 13-17, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2939672.2939768

1. INTRODUCTION

Measuring node proximity (i.e., similarity/relevance) on large scale networks is a fundamental building block in many application domains, ranging from computer vision [23], e-commerce [10, 9], social networks [28, 29], software engineering [27], disaster management [41] to biology [22] and epidemiology [31].

The state of the art has mainly focused on best leveraging the network topology for measuring node proximity. Among others, a prevalent choice for node proximity is random walk based methods (e.g., random walk with restart and many of its variants - see Section 5 for a review), largely due to its flexibility in summarizing multiple weighted relationships between nodes. These methods typically assume the input network is *given* a priori, with the *known*, whether static or dynamic, network topology and the associated edge weights. A few recent works aim to further infer the *optimal edge weights* based on the side information (e.g., user feedback, node/edge attribute information). Representative works include supervised random walk (SRW) [5] and learning to rank methods [2, 3]. Moreover, an implicit assumption behind these existing works is that *one* common (optimal) network, be it given a priori or learned by the algorithms, exists for all queries. Despite much progress has been made, several algorithmic questions have largely remained nascent.

Q1 *Optimal weights or optimal topology?* Both supervised random walks and learning to rank methods assume the topology of the input network is given and fixed. Yet, both the edge weights and the network topology could affect the proximity measures. More often than not, a noisy or missing edge could largely mislead random walks, and thus lead to sub-optimal proximity measures. On the other hand, almost any real-world network is incomplete and/or noisy [15]. How can we design an algorithm that is able to simultaneously learn the optimal network topology as well as the associated edge weights, to better measure node proximity?

Q2 *One-fits-all, or one-fits-one?* Most, if not all, of the existing works implicitly assume there exists *one* common optimal network for measuring node proximity for all the queries. From the optimization perspective, such a *one-(network)-fits-all-(queries)* assumption might be sub-optimal. How can we optimize the input network at the finer granularity, to learn a query-specific network for a given query (i.e., *one-(network)-fits-one-(query)*)?

Q3 *Offline learning or online learning?* Even if we restrict ourselves to only learning the optimal weights for only one single network, it is already a computationally intensive procedure. For example, both supervised random walks and learning to rank methods rely on a costly iterative sub-routine to compute a single gradient vector (a key step in these methods), and consequently the learning process has to be conducted in the off-line stage. The answers to Q1 and Q2 would further intensify the computational challenge. For Q1 (i.e., to learn the optimal network topology), it would require to significantly expand the parameter space to $O(n^2)$, where n is the number of the nodes in the network. For Q2 (i.e., to learn one network for one query), it would require to learn many networks (i.e., one network for one query), and to conduct learning in the on-line query stage. How can we design an effective on-line learning algorithm that simultaneously fulfills both Q1 and Q2, and in the meanwhile scales to billion-scale networks and responds in (near-)real time?

We aim to answer all these questions in this paper. First (*for Q1 - optimizaiton scope*), our proposed formulation explores a much larger parameter space, so that it is able to simultaneously infer the optimal network topology and the associated edge weights. Second (*for Q2 - optimization granularity*), while all the existing works assume there is one common optimal network for all queries, our method performs optimization at a much finer granularity, essentially being able to infer a (different) optimal network that is specific to a given query. Third (*for Q3 - optimization efficiency*), we carefully design our algorithms with a linear complexity wrt the neighborhood size of the user preference sets, which is often sub-linear wrt the size of the input network. Our main contributions can be summarized as follows:

- **Paradigm Shift:** we go beyond the two fundamental assumptions (i.e., Q1 and Q2) and introduce two new design objectives for node proximity measures, including (D1) to simultaneously learn the optimal topology and the associated edge weights, and (D2) to learn an optimal network for a given query node (i.e., one-network-fits-one-query).
- **Algorithms and Analysis:** we propose an optimization based approach to fulfill the two design objectives, and further develop a family of effective and efficient algorithms to solve it, including an on-line algorithm with a linear complexity wrt the neighborhood size of the user preference sets.
- **Empirical Evaluations:** we conduct extensive experiments on a diverse set of 10+ real networks and demonstrate that our algorithms (1) consistently outperform all existing methods on six commonly used evaluation metrics, (2) empirically scale *sub-linearly* to billion-scale networks and (3) respond in a fraction of a second.

The rest of the paper is organized as follows. Section 2 formally defines the query-specific optimal network learning problem. Section 3 introduces the proposed algorithms. Section 4 presents the empirical evaluation results. After reviewing related work in Section 5, we conclude the paper in Section 6.

Table 1: Symbols

Symbols	Definition
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$	a network
$\mathbf{A}$	adjacency matrix of the input network
$\mathbf{A}_s$	adjacency matrix of the learned optimal network
s	query node
$\mathbf{r}_s$	ranking vector for node s
$\mathcal{P} = \{x_1, x_2, \ldots, x_p\}$	set of positive nodes
$\mathcal{N} = \{y_1, y_2, \ldots, y_l\}$	set of negative nodes
$g(\cdot)$	loss function
n	number of nodes in the network
m	number of edges in the network
c	decay factor in random walks
b	margin in the loss function
λ, β	trade-off parameters

2. PROBLEM DEFINITION AND PRELIMINEARIES

In this section, we present the notations used throughout the paper (summarized in Table 1), formally define the query-specific optimal network learning problem and then give preliminaries on random walk based methods for node proximity.

2.1 Problem Definitions

We use bold upper-case letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for vectors (e.g., $\mathbf{v}$), and lowercase letters (e.g., α) for scalars. For matrix indexing, we use a convention similar to Matlab as follows. We use $\mathbf{A}(i, j)$ to denote the entry at the intersection of the i-th row and j-th column of matrix $\mathbf{A}$, $\mathbf{A}(i, :)$ to denote the i-th row of $\mathbf{A}$ and $\mathbf{A}(:, j)$ to denote the j-th column of $\mathbf{A}$. Besides, we use prime for matrix transpose (e.g., $\mathbf{A}'$ is the transpose of $\mathbf{A}$).

In our problem setting, we are given a network which is represented by an $n \times n$ normalized adjacency matrix, which has m non-zero elements (i.e., edges). As mentioned earlier, there is often rich information from user provided feedback/preference in some applications [32, 3]. For a user s, s/he could explicitly indicate some nodes that s/he wants to connect with, defined as *positive nodes*, and some other nodes that s/he wants to avoid, defined as *negative nodes*. We use the positive set $\mathcal{P} = \{x_1, x_2, \ldots, x_p\}$ to denote the set of positive nodes, i.e., s likes nodes x_i. Similarly, we use the negative set $\mathcal{N} = \{y_1, y_2, \ldots, y_l\}$ to denote the set of negative nodes, i.e., s dislikes y_i. See [5, 32] on how to select the positive and negative sets. Our goal is to learn an optimal network for this specific query node s, so that, when measured on the *learned network*, the proximities from s to the nodes in $\mathcal{P}$ and those in $\mathcal{N}$ match his/her preference. With these notations, the problem can be formally defined as follows:

PROBLEM 1. *Query-specific Optimal Network Learning*

Given: *a network with adjacency matrix* $\mathbf{A}$*, a query node* s*, positive node set* $\mathcal{P}$ *and negative node set* $\mathcal{N}$*.*

Learn: *an optimal network* $\mathbf{A}_s$ *specific to the query* s*.*

2.2 Preliminaries

Random walk based methods (such as random walk with restart [30] and many of its variants) have been a prevalent choice for proximity measures. Here, we present a brief summarization of random walk with restart (RWR), which is the base of our proposed methods. Please refer to Section 5 for detailed review of node proximity measures. For a given network $\mathbf{G}$, RWR works as follows. Consider a random surfer that starts from the node s. At each step, the random surfer has two options: (1) transmits to one of its neighbors with probability proportional to the edge weights; or (2) jumps back to the starting node s with a restart probability $(1-c)$. The proximity score from node s to node i is defined as the steady state probability r_{si} that the random surfer will visit node i. Define the ranking vector $\mathbf{r}_s$ for node s as the vector of proximity scores from node s to all the nodes on the network, RWR recursively computes the ranking vector as follows:

$$\mathbf{r}_s = c\mathbf{A}\mathbf{r}_s + (1-c)\mathbf{e}_s, \tag{1}$$

where $\mathbf{A}$ is the adjacency matrix of the network $\mathbf{G}$ and $\mathbf{e}_s$ is a vector of all zeros except 1 at the s-th position.

The ranking vector $\mathbf{r}_s$ can also be computed in the following closed form: $\mathbf{r}_s = (1-c)(\mathbf{I} - c\mathbf{A})^{-1}\mathbf{e}_s$. Let $\mathbf{Q} = (\mathbf{I} - c\mathbf{A})^{-1}$, we can see that $\mathbf{r}_s$ is the s-th column of $\mathbf{Q}$ with some constant scaling. In other words, we can regard the element $\mathbf{Q}(i,j)$ as the (scaled) proximity score from node j to node i. For the normalized matrix $\mathbf{A}$, it is often chosen as the stochastic matrix of the input network. Nonetheless, from the algorithmic perspective, we can also choose other forms to normalize $\mathbf{A}$, e.g., the so-called normalized graph laplacian [42]. In fact, as long as the leading eigenvalue of $\mathbf{A}$ is less than $1/c$, we can show that Eq. (1) always converges to its closed-form solution. Having this in mind, we will remove this constraint (i.e., $\mathbf{A}$ being a stochastic matrix) to simplify the proposed algorithms. We will also discuss how to impose such a constraint if the stochastic matrix is indeed a desired output in some applications.

3. PROPOSED ALGORITHMS

In this section, we present our algorithm, QUINT, to learn a QUery-specific optImal NeTwork (i.e. Problem 1). We first introduce the proposed formulations and give optimization solutions, followed up with scalable algorithms and some variants.

3.1 QUINT - Formulations

Given the input network with the adjacency matrix $\mathbf{A}$, and a query node s along with its associated positive node set $\mathcal{P}$ and negative node set $\mathcal{N}$, we want to learn an optimal network with adjacency matrix $\mathbf{A}_s$ such that the proximity from s to positive nodes in $\mathcal{P}$ and negative nodes in $\mathcal{N}$ matches the preference. The key ideas behind our proposed formulations can be summarized as follows. First, we want to avoid that the learned network $\mathbf{A}_s$ deviates too far from the input network $\mathbf{A}$. Second, on the *learned* network $\mathbf{A}_s$, the measured proximity should match the user preference. That is, if we compute the ranking vector $\mathbf{r}_s$ for s using the learned adjacency matrix $\mathbf{A}_s$, the proximity from s to any positive node in $\mathcal{P}$ is greater than to any negative node in $\mathcal{N}$, i.e., $\mathbf{r}_s(x) > \mathbf{r}_s(y), \forall x \in \mathcal{P}, \forall y \in \mathcal{N}$. Based on the above intuition, we propose the following formulation to learn a query-specific network:

$$\begin{aligned} \arg\min_{\mathbf{A}_s} \quad & \|\mathbf{A}_s - \mathbf{A}\|_F^2 \\ \text{s.t.,} \quad & \mathbf{Q}(x,s) > \mathbf{Q}(y,s), \forall x \in \mathcal{P}, \forall y \in \mathcal{N}, \end{aligned} \tag{2}$$

where $\mathbf{Q} = (\mathbf{I} - c\mathbf{A}_s)^{-1}$ contains the pairwise node proximities on the learned network. In Eq. (2), the objective function states that the distance from the learned network adjacency matrix $\mathbf{A}_s$ to the original network $\mathbf{A}$ measured by Frobenuis norm should be minimized; while in the constraint, we want the proximity from node s to any positive node in $\mathcal{P}$ to be greater than to any negative node in $\mathcal{N}$. This is a hard constraint as we do not allow any exception. In practice, the constraint might not be satisfied by all the possible pairs of positive and negative nodes. Instead, we can relax the constraint and introduce some penalization if violations occur. This relaxed soft version can be formulated as follows:

$$\arg\min_{\mathbf{A}_s} \mathcal{L}(\mathbf{A}_s) = \lambda\|\mathbf{A}_s - \mathbf{A}\|_F^2 + \sum_{x\in\mathcal{P}, y\in\mathcal{N}} g(\mathbf{Q}(y,s) - \mathbf{Q}(x,s)), \tag{3}$$

where λ is the trade-off parameter that balances between the adjacency matrices difference and constraint violations. To penalize the constraint violations, we introduce the loss function $g(\cdot)$. Ideally, we want $\mathbf{Q}(y,s) < \mathbf{Q}(x,s)$. Hence, if $\mathbf{Q}(y,s) < \mathbf{Q}(x,s)$, i.e., the constraint is not violated, then $g(\cdot) = 0$; otherwise if $\mathbf{Q}(y,s) > \mathbf{Q}(x,s)$, then $g(\cdot) > 0$ to penalize such a violation. In the paper, we consider the Wilcoxon-Mann-Whitney (WMW) loss [35] with width b, which is differentiable and was originally proposed to optimize the area under the ROC curve (AUC):

$$g(x) = \frac{1}{1 + exp(-x/b)}.$$

Remarks: The formulation in Eq. (3) bears a high-level resemblance to the supervised random walks (SRW) [5] in terms of the way they encode the user preference. Nonetheless, there are several subtle differences between them. First, we explore a much larger parameter space in the order of $O(n^2)$, whereas SRW only searches in a d-dimensional vector space, where d is the length of the feature vector. The potential benefit is that we are able to search for both the optimal topology and the associated edge weights. Second, our formulation is tailored for a specific query node, and thus is potentially able to learn an optimal network for that specific query (instead of one universal network for all the queries). Third, in our formulation, we drop the typical constraint that requires the learned network $\mathbf{A}_s$ to be a stochastic matrix. We find that such a relaxation will greatly ease the optimization algorithm, without a noticeable empirical performance degradation. For readers who are interested in keeping this constraint in our formulation, we will present a variant to do so in Section 3.4.3.

3.2 QUINT - Optimization Solutions

The objective function in Eq. (3) is non-convex. We aim to solve it using gradient descent based methods by first calculating the derivative of $\mathcal{L}(\mathbf{A}_s)$ w.r.t. $\mathbf{A}_s$ and then updating the adjacency matrix $\mathbf{A}_s$ along the negative direction of the derivative.

The derivative of $\mathcal{L}(\mathbf{A}_s)$ w.r.t. $\mathbf{A}_s$ can be written as fol-

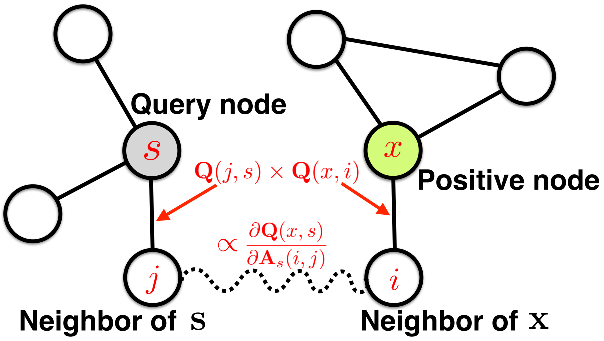

Query node
s
x
Positive node
$\mathbf{Q}(j,s) \times \mathbf{Q}(x,i)$
$\propto \frac{\partial \mathbf{Q}(x,s)}{\partial \mathbf{A}_s(i,j)}$
j
i
Neighbor of s
Neighbor of x

Figure 1: Update on $\mathbf{A}_s(i,j)$.

lows:

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{A}_s)}{\partial \mathbf{A}_s} &= 2\lambda(\mathbf{A}_s - \mathbf{A}) + \sum_{x\in\mathcal{P},y\in\mathcal{N}} \frac{\partial g(\mathbf{Q}(y,s)-\mathbf{Q}(x,s))}{\partial \mathbf{A}_s} \\ &= 2\lambda(\mathbf{A}_s - \mathbf{A}) + \sum_{x,y} \frac{\partial g(d_{yx})}{\partial d_{yx}}\left(\frac{\partial \mathbf{Q}(y,s)}{\partial \mathbf{A}_s} - \frac{\partial \mathbf{Q}(x,s)}{\partial \mathbf{A}_s}\right),\end{aligned} \tag{4}$$

where we denote $d_{yx} = \mathbf{Q}(y,s) - \mathbf{Q}(x,s)$ and we apply chain rule in the second step. The WMW loss function is differentiable and its derivative is computed as: $\frac{\partial g(d_{yx})}{\partial d_{yx}} = \frac{1}{b}g(d_{yx})(1-g(d_{yx}))$. To compute the derivative of $\frac{\partial \mathbf{Q}(y,s)}{\partial \mathbf{A}_s}$ is more involved, since $\mathbf{Q}$ is an inverse of a matrix, which itself is a function of $\mathbf{A}_s$.

From the basic identity for derivatives of a matrix inverse [24], we have that:

$$\frac{\partial \mathbf{Q}}{\partial \mathbf{A}_s(i,j)} = -\mathbf{Q}\frac{\partial (\mathbf{I}-c\mathbf{A}_s)}{\partial \mathbf{A}_s(i,j)}\mathbf{Q} = c\mathbf{Q}\mathbf{J}^{ij}\mathbf{Q}, \tag{5}$$

where $1 \le i \le n$, $1 \le j \le n$, and $\mathbf{J}^{i,j}$ is a single-entry matrix with $\mathbf{J}(i,j) = 1$ and zeros everywhere else.

Based on the above equation, we have that

$$\frac{\partial \mathbf{Q}(x,s)}{\partial \mathbf{A}_s(i,j)} = c\mathbf{Q}(x,i)\mathbf{Q}(j,s). \tag{6}$$

The intuition behind the formulation is that the update to $\mathbf{A}_s(i,j)$ is proportional to the product of the proximity from i to x and the proximity from s to j. In other words, if i is a close neighbor of x (i.e., large $\mathbf{Q}(x,i)$) and j is a close neighbor of s (i.e., large $\mathbf{Q}(j,s)$), as illustrated in Figure 1, then $\mathbf{A}_s(i,j)$ will have a relatively large update. This makes sense since a larger increase on $\mathbf{A}_s(i,j)$ (i.e., a pair of close neighbors of the query node s and a positive node x, respectively) will increase the chance of reaching x from s by random walks (i.e., increasing $\mathbf{Q}(x,s)$ measured on the updated adjacency matrix $\mathbf{A}_s$).

Following this, we can compute the derivative of $\frac{\partial \mathbf{Q}(x,s)}{\partial \mathbf{A}}$ as:

$$\frac{\partial \mathbf{Q}(x,s)}{\partial \mathbf{A}_s} = c[\mathbf{Q}(x,i)\mathbf{Q}(j,s)]_{1\le i\le n, 1\le j\le n} = c\mathbf{Q}(x,:)'\mathbf{Q}(:,s)'. \tag{7}$$

Similarly, the derivative of $\frac{\partial \mathbf{Q}(y,s)}{\partial \mathbf{A}_s}$ can be computed as:

$$\frac{\partial \mathbf{Q}(y,s)}{\partial \mathbf{A}_s} = c\mathbf{Q}(y,:)'\mathbf{Q}(:,s)'. \tag{8}$$

Therefore, the derivative of $\mathcal{L}(\mathbf{A}_s)$ w.r.t. $\mathbf{A}_s$ can be rewritten as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{A}_s)}{\partial \mathbf{A}_s} &= 2\lambda(\mathbf{A}_s - \mathbf{A}) + \sum_{x,y} \frac{\partial g(d_{yx})}{\partial d_{yx}}\left(\frac{\partial \mathbf{Q}(y,s)}{\partial \mathbf{A}_s} - \frac{\partial \mathbf{Q}(x,s)}{\partial \mathbf{A}_s}\right) \\ &= 2\lambda(\mathbf{A}_s - \mathbf{A}) + \sum_{x,y} c\frac{\partial g(d_{yx})}{\partial d_{yx}}(\mathbf{Q}(y,:)' - \mathbf{Q}(x,:)')\mathbf{Q}(:,s)'.\end{aligned} \tag{9}$$

The above optimization solution for the query-specific network learning is summarized in Algorithm 1. From the algorithm we can see that, we not only update the weights of existing edges, but might also add an unseen edge and/or remove a noisy edge during the update (i.e., changing the original topology).

Algorithm 1 QUINT– Learning a Query-Specific Optimal Network

```
Input: (1) the initial network adjacency matrix $\mathbf{A}$,
    (2) a query node $s$,
    (3) a positive node set $\mathcal{P}$
    (4) a negative node set $\mathcal{N}$
    (5) parameters $c$, $\lambda$, $b$, and step size $\eta$.
Output: The adjacency matrix $\mathbf{A}_s$ of the optimal network
    specific to query $s$.
 1: Initialize $\mathbf{A}_s = \mathbf{A}$;
 2: while not converged do
 3:   for each positive node $x$ from $\mathcal{P}$ do
 4:     for each negative node $y$ from $\mathcal{N}$ do
 5:       Compute $\mathbf{Q}(:,s)$, $\mathbf{Q}(y,:)$ and $\mathbf{Q}(x,:)$;
 6:       Compute $c\frac{\partial g(d_{yx})}{\partial d_{yx}}(\mathbf{Q}(y,:)' - \mathbf{Q}(x,:)')\mathbf{Q}(:,s)'$;
 7:     end for
 8:   end for
 9:   Compute the derivative $\frac{\partial \mathcal{L}(\mathbf{A}_s)}{\partial \mathbf{A}_s}$ by Eq (9);
10:   Update $\mathbf{A}_s = \mathbf{A}_s - \eta \cdot \frac{\partial \mathcal{L}(\mathbf{A}_s)}{\partial \mathbf{A}_s}$;
11: end while
12: return the learned network adjacency matrix $\mathbf{A}_s$;
```

We summarize complexities of Algorithm 1 in Theorem 1.

Theorem 1. *(Time and Space Complexities of Algorithm 1). Algorithm 1 takes $O(T_1|\mathcal{P}|\cdot|\mathcal{N}|(T_2m+n^2))$ time, where T_1 is the number of iterations to convergence, and T_2 is the number of iterations in the power method for computing ranking vector. It takes additional $O(n^2)$ space.*

Proof. The inner loop, line 5 and line 6, is executed $|\mathcal{P}|\cdot|\mathcal{N}|$ times. For line 5, it would be too expensive ($O(n^3)$) to first compute $\mathbf{Q} = (\mathbf{I} - c\mathbf{A}_s)^{-1}$. Instead, we can use power method to extract the corresponding column of $\mathbf{Q}$ by Eq. (1) and this line would take $O(T_2m)$ time, where T_2 is the number of iterations in the power method and m is the number of edges. Line 6 involves a multiplication of a column vector and a row vector, which would take $O(n^2)$. As a result, the overall time complexity is $O(T_1|\mathcal{P}|\cdot|\mathcal{N}|(T_2m+n^2))$, where T_1 is the number of iterations to convergence. The two vectors in the multiplication are dense and their multiplication would result in an $n \times n$ dense matrix, taking $O(n^2)$ additional space. □

3.3 QUINT - Scale-up for Online Queries

To learn an optimal network for one query using Algorithm 1, it would take $O(T_1|\mathcal{P}|\cdot|\mathcal{N}|(T_2m+n^2))$, which is not scalable to large real network data (usually in the order of millions or billions of nodes/edges), not to mention that it practically eliminates the possibility to conduct learning in on-line stage. In this subsection, we introduce an effective on-line algorithm to learn the optimal network for each query. The key idea behind the fast solution is that the optimal network can be approximated by a rank-one perturbation to the original network. If we observe more closely at the derivative of $\mathcal{L}(\mathbf{A}_s)$ w.r.t. $\mathbf{A}_s$ in Eq. (9), the second term (the summation over x and y) is exactly a matrix of rank one for a specific query.

Following the rank-one perturbation assumption, we can approximate the optimal network as $\mathbf{A}_s = \mathbf{A} + \mathbf{f}\mathbf{g}'$, where $\mathbf{f}$ and $\mathbf{g}$ are n-dimensional vectors we want to learn. Having this, we can formulate our optimal network learning with rank-one perturbation as follows:

$$\begin{aligned} \arg\min_{\mathbf{f},\mathbf{g}} \mathcal{L}(\mathbf{f},\mathbf{g}) &= \lambda\|\mathbf{f}\mathbf{g}'\|_F^2 + \beta(\|\mathbf{f}\|^2 + \|\mathbf{g}\|^2) \\ &+ \sum_{x\in\mathcal{P},y\in\mathcal{N}} g(\mathbf{Q}(y,s) - \mathbf{Q}(x,s)), \end{aligned} \tag{10}$$

where λ and β are the trade-off parameters. Notice that in this formulation, $\mathbf{Q} = (\mathbf{I} - c\mathbf{A} - c\mathbf{f}\mathbf{g}')^{-1}$ is a function of both $\mathbf{f}$ and $\mathbf{g}$.

We apply an alternating strategy to solve the above formulation. Let us first fix $\mathbf{g}$ and solve for $\mathbf{f}$. The derivative of $\mathcal{L}(\mathbf{f})$ w.r.t. $\mathbf{f}$ can be written as follows:

$$\begin{aligned} \frac{\partial\mathcal{L}(\mathbf{f})}{\partial\mathbf{f}} &= 2\lambda\mathbf{f}\mathbf{g}'\mathbf{g} + 2\beta\mathbf{f} + \sum_{x\in\mathcal{P},y\in\mathcal{N}} \frac{\partial g(\mathbf{Q}(y,s)-\mathbf{Q}(x,s))}{\partial\mathbf{f}} \\ &= 2\lambda\mathbf{f}\mathbf{g}'\mathbf{g} + 2\beta\mathbf{f} + \sum_{x,y} \frac{\partial g(d_{yx})}{\partial d_{yx}}\left(\frac{\partial\mathbf{Q}(y,s)}{\partial\mathbf{f}} - \frac{\partial\mathbf{Q}(x,s)}{\partial\mathbf{f}}\right), \end{aligned} \tag{11}$$

where we denote $d_{yx} = \mathbf{Q}(y,s) - \mathbf{Q}(x,s)$. The question now becomes how to compute the derivative $\frac{\partial\mathbf{Q}(x,s)}{\partial\mathbf{f}}$.

Again, according to the basic identity for derivative of a matrix inverse, we have the following:

$$\frac{\partial\mathbf{Q}}{\partial\mathbf{f}(i)} = -\mathbf{Q}\frac{\partial(\mathbf{I} - c\mathbf{A} - c\mathbf{f}\mathbf{g}')}{\partial\mathbf{f}(i)}\mathbf{Q} = c\mathbf{Q}\mathbf{e}_i\mathbf{g}'\mathbf{Q}. \tag{12}$$

Following this, we obtain

$$\frac{\partial\mathbf{Q}(x,s)}{\partial\mathbf{f}(i)} = c\mathbf{Q}(x,i)\mathbf{Q}(:,s)'\mathbf{g}. \tag{13}$$

Now, we can compute the derivative $\frac{\partial\mathbf{Q}(x,s)}{\partial\mathbf{f}}$ as follows:

$$\frac{\partial\mathbf{Q}(x,s)}{\partial\mathbf{f}} = c[\mathbf{Q}(x,i)\mathbf{Q}(:,s)'\mathbf{g}]_{1\leq i\leq n} = c\mathbf{Q}(x,:)'\mathbf{Q}(:,s)'\mathbf{g}. \tag{14}$$

Therefore, the derivative of $\mathcal{L}(\mathbf{f})$ w.r.t. $\mathbf{f}$ can be computed as follows:

$$\begin{aligned} \frac{\mathcal{L}(\mathbf{f})}{\partial\mathbf{f}} &= 2\lambda\mathbf{f}\mathbf{g}'\mathbf{g} + 2\beta\mathbf{f} + \sum_{x,y} \frac{\partial g(d_{yx})}{\partial d_{yx}}\left(\frac{\partial\mathbf{Q}(y,s)}{\partial\mathbf{f}} - \frac{\partial\mathbf{Q}(x,s)}{\partial\mathbf{f}}\right) \\ &= 2\lambda\mathbf{f}\mathbf{g}'\mathbf{g} + 2\beta\mathbf{f} + \sum_{x,y} c\frac{\partial g(d_{yx})}{\partial d_{yx}}(\mathbf{Q}(y,:)' - \mathbf{Q}(x,:)')(\mathbf{Q}(:,s)'\mathbf{g}). \end{aligned} \tag{15}$$

The computation for the derivative of $\mathcal{L}(\mathbf{g})$ w.r.t. $\mathbf{g}$ is similar to Eq. (15) with $\mathbf{f}$ substituted with $\mathbf{g}$ and $\mathbf{g}$ substituted with $\mathbf{f}$.

We summarize the above optimization solution for optimal network learning with rank-one perturbation in Algorithm 2, along with its complexity analysis in Theorem 2.

Theorem 2. *(Time and Space Complexities of Algorithm 2). Algorithm 2 takes $O(T_1|\mathcal{P}|\cdot|\mathcal{N}|(T_2m+n))$ time, where T_1 is the number of iterations to convergence, and T_2 is the number of iterations in the power method for computing ranking vector. It takes additional $O(n)$ space.*

Algorithm 2 QUINT-rankOne – Learning a Query-Specific Optimal Network

```
Input: (1) the initial network adjacency matrix $\mathbf{A}$,
    (2) a query node $s$,
    (3) a positive node set $\mathcal{P}$
    (4) a negative node set $\mathcal{N}$
    (5) parameters $c$, $\lambda$, $\beta$, $b$, and step size $\eta$.
Output: The rank-one perturbation to the network $\mathbf{f}$ and $\mathbf{g}$.
 1: Initialize $\mathbf{f}$ and $\mathbf{g}$;
 2: while not converged do
 3:   for each positive node $x$ from $\mathcal{P}$ do
 4:     for each negative node $y$ from $\mathcal{N}$ do
 5:       Compute $\mathbf{Q}(:,s)$, $\mathbf{Q}(y,:)$ and $\mathbf{Q}(x,:)$;
 6:       Compute $c\frac{\partial g(d_{yx})}{\partial d_{yx}}(\mathbf{Q}(y,:)' - \mathbf{Q}(x,:)')(\mathbf{Q}(:,s)'\mathbf{g})$;
 7:     end for
 8:   end for
 9:   Compute the derivative $\frac{\partial\mathcal{L}(\mathbf{f})}{\partial\mathbf{f}}$ by Eq (15);
10:   Compute the derivative $\frac{\partial\mathcal{L}(\mathbf{g})}{\partial\mathbf{g}}$;
11:   Update $\mathbf{f} = \mathbf{f} - \eta\cdot\frac{\partial\mathcal{L}(\mathbf{f})}{\partial\mathbf{f}}$;
12:   Update $\mathbf{g} = \mathbf{g} - \eta\cdot\frac{\partial\mathcal{L}(\mathbf{g})}{\partial\mathbf{g}}$;
13: end while
14: return the learned rank-one perturbations $\mathbf{f}$ and $\mathbf{g}$;
```

Proof. The inner loop, line 5 and line 6, is executed $|\mathcal{P}|\cdot|\mathcal{N}|$ times. For line 5, we can use power method to extract the corresponding column of $\mathbf{Q}$ by Eq. (1) and this line would take $O(T_2m)$ time, where T_2 is the number of iterations in the power method. Line 6 takes $O(n)$. As a result, the overall time complexity is $O(T_1|\mathcal{P}|\cdot|\mathcal{N}|(T_2m+n))$, where T_1 is the number of iterations to convergence. The additional space takes $O(n)$, i.e., the length of the vectors. □

Remarks: The major computational overhead of Algorithm 2 comes from line 5 to extract certain columns of $\mathbf{Q}$, which leads to an $O(T_2m)$ complexity. In the next subsection, we propose additional ways to further speed up the algorithm to scale linearly and even sub-linearly on n.

3.4 QUINT - Variants

In this subsection, we first provide several ways to further speed up the proposed algorithm to scale linearly and even sub-linearly; and then point out a way to satisfy stochasticity of the output and to learn the decay factor c in RWR.

3.4.1 Variant #1: Taylor Approximation for $\mathbf{Q}$

As we mentioned above, the major time complexity of Algorithm 2 is due to the extraction of certain columns of $\mathbf{Q}$ using the power method. In fact, we could approximate $\mathbf{Q}$ using Taylor approximation as follows:

$$\begin{aligned} \mathbf{Q} &= (\mathbf{I} - c\mathbf{A})^{-1} \\ &\approx \mathbf{I} + \textstyle\sum_{i=1}^{k} c^k\mathbf{A}^k. \end{aligned} \tag{16}$$

If we use first order Taylor approximation, i.e. $k = 1$ in Eq. (16), line 5 in Algorithm 2 would only take $O(n)$ instead of $O(T_2m)$. Therefore, the overall time complexity of Algorithm 2 can be reduced to $O(T_1|\mathcal{P}|\cdot|\mathcal{N}|n)$.

3.4.2 Variant #2: Localized Rank-One Perturbation

It is often not necessary to update the global adjacency matrix, instead, we could focus on certain local zones in the network that will play a more important role to the proximities from the query node s.

In particular, we would update the local structure in the neighborhood of the query node as well as the neighborhood of the preference sets. Denote the neighborhood of node s (including s) by $\mathbb{N}(s) = \{z|(z,s) \in \mathbf{E}\} \cup \{s\}$ and denote the neighborhood of the positive and negative nodes by $\mathbb{N}(\mathcal{P},\mathcal{N}) = \{z|z \in \mathbb{N}(x) \text{ or } z \in \mathbb{N}(y), \forall x \in \mathcal{P}, \forall y \in \mathcal{N}\}$. When we update $\mathbf{f}$ using Eq. (15), we only need to update $\mathbf{f}(i), i \in \mathbb{N}(s)$. Similarly, when we update $\mathbf{g}$, we only need to update $\mathbf{g}(i), i \in \mathbb{N}(\mathcal{P},\mathcal{N})$. This will further bring down the time complexity of Algorithm 2 to $O(T_1|\mathcal{P}| \cdot |\mathcal{N}| \max(|\mathbb{N}(s)|, |\mathbb{N}(\mathcal{P},\mathcal{N})|))$ (Theorem 3).

THEOREM 3. *(Time Complexity of QUINT-rankOne with First-order Taylor Approximation and Localized Rank-One Perturbation.) If we use first-order Taylor approximation for $\mathbf{Q}$ and localized rank-one perturbation in QUINT-rankOne, it would take $O(T_1|\mathcal{P}| \cdot |\mathcal{N}| \max(|\mathbb{N}(s)|, |\mathbb{N}(\mathcal{P},\mathcal{N})|))$ to learn the optimal network.*

PROOF. Omitted for brevity. □

3.4.3 Variant #3: Stochastic Matrix $\mathbf{A}_s$

As mentioned earlier, we do not require the learned network $\mathbf{A}_s$ to be a stochastic matrix, which largely eases the optimization process. If a stochastic matrix is indeed the desired output network, we can naturally modify the proposed algorithms to fulfill it. To be specific, immediately following the gradient descent step in the proposed algorithms, we introduce a *simplex projection* operation for each column of $\mathbf{A}_s$ [33]. In this way, we ensure that matrix $\mathbf{A}_s$ is always a valid stochastic matrix. However, we do not observe a noticeable empirical improvement by the simplex projection operation, yet it introduces an additional $O(n^2 \log n)$ (since we need to do so for n columns) into the overall time complexity. Therefore, we do not recommend it in practice.

3.4.4 variant #4: Learning Decay Factor in RWR

An important parameter in RWR is the decay factor c, which is usually manually set. As a side product, our proposed methods naturally provide a way to learn the parameter c. In this setting, we assume the network structure $\mathbf{A}$ is known and fixed, and we have the following optimization formulation:

$$\arg\min_c \mathcal{L}(c) = \sum_{x \in \mathcal{P}, y \in \mathcal{N}} g(\mathbf{Q}(y,s) - \mathbf{Q}(x,s)).$$

The key to the above optimization problem is to calculate the derivative of $\mathcal{L}(c)$ w.r.t. c. We have that

$$\begin{aligned} \tfrac{\partial \mathcal{L}(c)}{\partial c} &= \sum_{x \in \mathcal{P}, y \in \mathcal{N}} \tfrac{\partial g(\mathbf{Q}(y,s) - \mathbf{Q}(x,s))}{\partial c} \\ &= \sum_{x,y} \tfrac{\partial g(d_{yx})}{\partial d_{yx}} (\tfrac{\partial \mathbf{Q}(y,s)}{\partial \mathbf{c}} - \tfrac{\partial \mathbf{Q}(x,s)}{\partial \mathbf{c}}). \end{aligned}$$

Again, we have the following identity

$$\frac{\partial \mathbf{Q}}{\partial c} = -\mathbf{Q}\frac{\partial(\mathbf{I} - c\mathbf{A})}{\partial c}\mathbf{Q} = \mathbf{QAQ}.$$

Following this, we can get

$$\frac{\partial \mathbf{Q}(x,s)}{\partial c} = \mathbf{Q}(x,:)\mathbf{AQ}(:,s).$$

The derivative of $\mathcal{L}(c)$ w.r.t. c becomes

$$\begin{aligned} \tfrac{\partial \mathcal{L}(c)}{\partial c} &= \sum_{x,y} \tfrac{\partial g(d_{yx})}{\partial d_{yx}} (\tfrac{\partial \mathbf{Q}(y,s)}{\partial \mathbf{c}} - \tfrac{\partial \mathbf{Q}(x,s)}{\partial \mathbf{c}}) \\ &= \sum_{x,y} \tfrac{\partial g(d_{yx})}{\partial d_{yx}} (\mathbf{Q}(y,:) - \mathbf{Q}(x,:))\mathbf{AQ}(:,s). \end{aligned}$$

We omit the detailed algorithm description for learning the parameter c due to the space limit.

4. EMPIRICAL EVALUATIONS

In this section, we design and conduct experiments mainly to answer the following questions:

- *Effectiveness:* How effective are the proposed algorithms for learning a query-specific optimal network?
- *Efficiency:* How fast and scalable are the proposed algorithms?

4.1 Datasets

We test our algorithms on a diverse set of real-world network datasets, including collaboration networks, social networks, infrastructure networks, etc. The statistics of all the datasets used are summarized in Table 2.

COLLABORATION NETWORKS. We use four collaboration networks from arXiv preprint archive[1]. In the networks, the nodes are authors and an edge exists between two authors if they have co-authored the same paper. We consider such collaboration networks from four areas of Physics: Astrophysics (*Astro-Ph*), general relativity and quantum cosmology (*GR-QC*), high energy physics theory (*Hep-TH*) and high energy physics phenomenology (*Hep-PH*).

SOCIAL NETWORKS. Here, nodes are users and edges indicate social relationships. Among them, *Last.fm* provides a music streaming and recommendation service and users can befriend with each other. *LiveJournal* provides social networking service where users can write a blog, journal or diary. *LinkedIn* provides social networking service to professionals and helps people find the right position. *Twitter* is a popular micro-blogging website where people can follow each other. *Email-Enron* is a communication network that covers around half million email communications [17]. The nodes are email addresses and an edge exists between two nodes if they have communicated through emails.

INFRASTRUCTURE NETWORKS. *Oregon* is a network of routers in Autonomous Systems (AS) inferred from Oregon route-views between March 31, 2001 and May 26, 2001 [17]. *Airport* network represents one month of internal US air traffic links between 2,833 airports[2].

SPORTS NETWORKS. *NBA* dataset contains NBA and ABA statistics from the year of 1946 to the year of 2009 [18]. The nodes are players and two players have an edge if they played in the same team before.

BIOLOGY NETWORKS. *Gene* is a human gene regulatory network obtained based on gene expression profiles [3]. *Protein* is a network of proteins obtained by BLAST algorithm [4] for comparing the sequence similarity.

4.2 Experiment Setup

We test the effectiveness of our query-specific optimal network learning algorithms in the task of *link prediction* – to

[1] http://arxiv.org/

[2] http://www.levmuchnik.net/Content/Networks/NetworkData.html

[3] http://www.cise.ufl.edu/research/sparse/matrices/Belcastro/human_gene2.html

Table 2: Statistics of the datasets.

Category	Network	# Nodes	# Edges
Collaboration	Astro-Ph	19,144	198,110
	GR-QC	5,242	14,496
	Hep-TH	10,700	25,997
	Hep-PH	12,527	118,515
Social	Email-Enron	36,692	183,831
	Last.fm	136,420	1,685,524
	LiveJournal	3,017,286	87,037,567
	LinkedIn	6,726,011	19,360,690
	Twitter	40,171,624	1,468,365,182
Infrastructure	Oregon	7,352	15,665
	Airport	2,833	7,602
Sports	NBA	3,924	126,994
Biology	Gene	14,340	43,588
	Protein	2,712	25,979

predict links in the network that will be created in the future. We reserve half of the edges as the training set (i.e., the observed network) and the rest as testing set and for choosing the positive and negative node sets. In particular, for each query node s, we choose 5 positive nodes to which it has an edge and 5 negative nodes to which it has no edge. To simulate the noisy edges, we add edges from the query node to its negative nodes with weight 1 and edges from the neighborhood of the query node to the neighborhood of the negative nodes with weight 0.1.

Evaluation metrics: We quantify the performance for link prediction using several metrics. Among them, Mean Average Precision (**MAP**) [36] measures the overall performance based on precision at different recall levels. Mean Percentage Ranking (**MPR**) [13] computes the average percentile-ranking over all node pairs (e.g., a percentile-ranking 0% means that connected node has the highest ranking score). Half-Life Utility (**HLU**) [8] estimates how likely a node will connect to the nodes in the ranking list, and the likelihood would decay exponentially as the rank increases.

In addition, we also consider the commonly used Area under the ROC curve (**AUC**), Precision@K, and Recall@K. Ideally, we want to achieve lower MPR value and higher values on other metrics.

Repeatability of Experimental Results. All the datasets are publicly available. We will release the code of the proposed algorithms through authors' website. For all the results reported, we set $\lambda = \beta = 0.1$ and $b = 1$. The experiments are performed on a Windows machine with four 3.5GHz Intel Cores and 256GB RAM.

4.3 Effectiveness Results

We perform the effectiveness comparisons with the following methods:

1. Random Walk with Restart (RWR): perform RWR on the network with no side information used.
2. Common Neighbors: count the number of common neighbors as ranking scores.
3. Adamic/Adar [1]: similar to Common Neighbors but with more weights given to those common neighbors with fewer degrees.
4. Supervised Random Walk (SRW) [5]: learn a function that assigns weights to edges. For an edge, we use its two nodes' degrees and number of their common neighbors as features.
5. wiZAN_Dual [36]: incorporate node similarities into one-class collaborative filtering.
6. ProSIN [32]: refine the network structure in response to the user feedback.
7. QUINT-Basic: use the power method to extract columns of **Q** in Algorithm 1.
8. QUINT-Basic1st: use first order Taylor approximation of **Q** for the columns extraction in Algorithm 1.
9. QUINT-rankOne: use first order Taylor approximation of **Q** for the columns extraction in Algorithm 2.

The effectiveness comparison results across a diverse set of networks on the six evaluation metrics are shown from Figure 2 to Figure 6. We have the following observations: (1) our QUINT family algorithms consistently outperform other comparison methods across all the datasets on all the six evaluation metrics. For example, on the *Astro* dataset, compared with the best competitor ProSIN, QUINT-rankOne is 21.4% higher on MAP, 13.5% higher on HLU. (2) The QUINT-Basic1st and QUINT-rankOne share a similar performance as QUINT-Basic, indicating that both approximation strategies are effective.

We also perform a t-test between the MAP results of QUINT-rankOne and the best competitor ProSIN on *Astro*. The p-value is 3.7e-44, which suggests the improvement of the proposed methods is indeed statistically significant.

4.4 Efficiency Results

Scalability: We show the running time per query vs. number of nodes (n) and edges (m) on the *LinkedIn*, *LiveJournal* and *Twitter* networks from Figure 8 to Figure 13. We do not report the running time of QUINT-Basic on *Twitter* dataset because of "out of memory" error ($O(n^2)$ space complexity). Our QUINT-rankOne algorithm scales sub-linearly w.r.t. to both n and m on *LinkedIn* and *Twitter*, while scales linearly w.r.t. to n and m on *LiveJournal*. This is in accordance with our complexity analysis, since QUINT-rankOne is linear w.r.t. to the size of neighborhood of the query as well as positive and negative nodes, which is *at most* linear w.r.t. n and m. In addition, it only takes $\sim$ 0.34s for QUINT-rankOne to process one query on the largest dataset (i.e. *Twitter*) with $\sim$ 1.4 billion edges and $\sim$ 40 million nodes.

5. RELATED WORK

In this section, we review the related work.

Node Proximity and RWR. Measuring node proximity in a network is an important task in many real applications, ranging from computer vision [23], e-commerce [10, 9], social networks [28, 29, 11], software engineering [27], disaster management [41] to biology [22] and epidemiology [31]. The state of the art has mostly focused on best exploring the topology information for measuring the proximity. Among others, the random walk based methods have gained prevalence largely due to its superiority in capturing the multiple weighted relationships between nodes. Despite their successes, several limitations still exist.

First (Q1), most of the existing works assume a fixed network topology along with its associated edge weights, although the real networks could be incomplete and noisy. For this reason, Agarwal et al. [3, 2] propose to compute the global rankings of nodes in a network by learning the edge weights; Backstrom and Leskovec [5] propose to learn the optimal edge weights based on the node and edge attributes/features. However, these methods still assume the topology of the given network is fixed. Another work proposes to use side information to refine the network topology

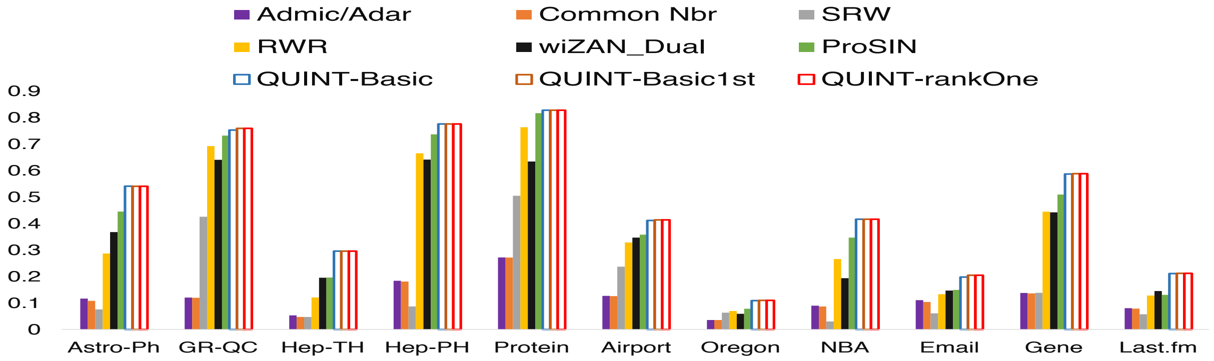


Figure 2: MAP performance comparisons on different network datasets. Higher is better. Best viewed in color.

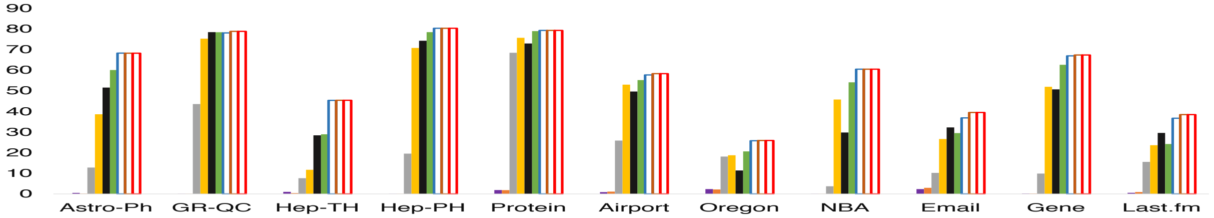


Figure 3: HLU performance comparisons on different network datasets. Higher is better. Best viewed in color.

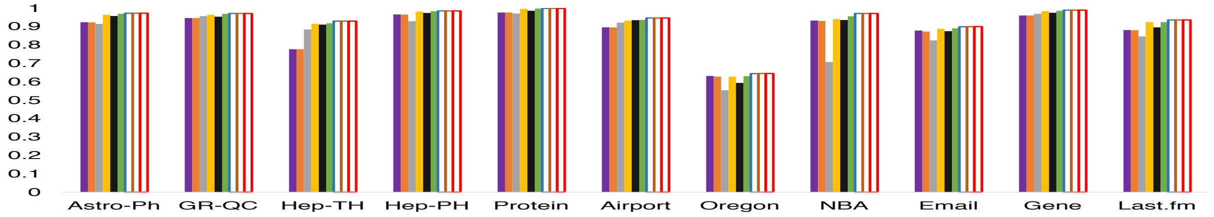


Figure 4: AUC performance comparisons on different network datasets. Higher is better. Best viewed in color.

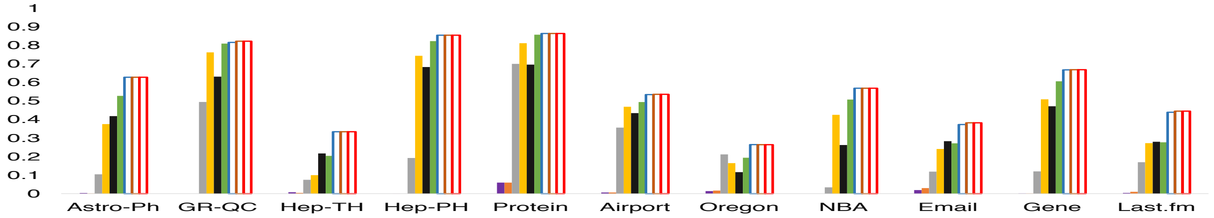


Figure 5: Precision@20 performance comparisons on different network datasets. Higher is better. Best viewed in color.

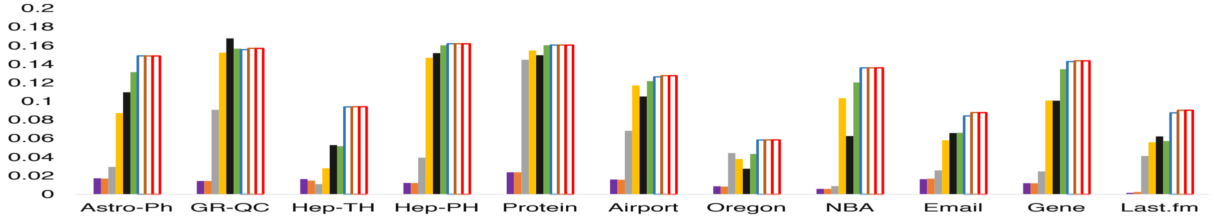


Figure 6: Recall@5 performance comparisons on different network datasets. Higher is better. Best viewed in color.

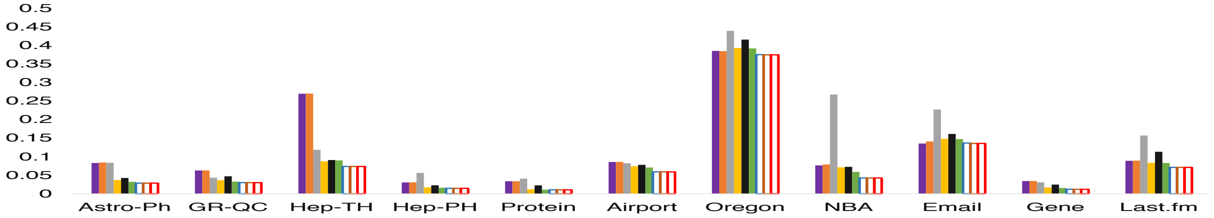
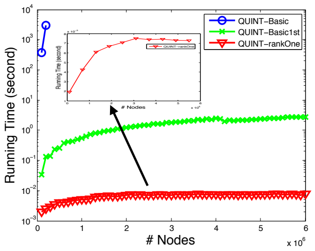


Figure 7: MPR performance comparisons on different network datasets. **Lower** is better. Best viewed in color.



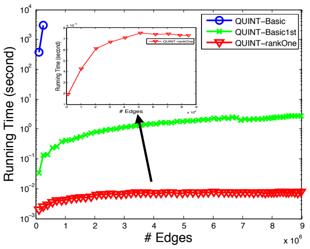
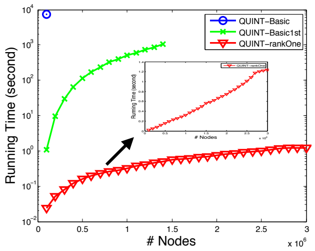


Figure 8: Running time per query vs. number of nodes on LinkedIn dataset (y-axis is in log scale). Inset: scalability of QUINT-rankOne only (y-axis is in linear scale).

Figure 9: Running time per query vs. number of edges on LinkedIn dataset (y-axis is in log scale). Inset: scalability of QUINT-rankOne only (y-axis is in linear scale).



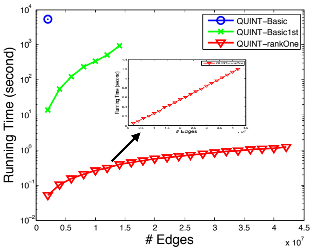
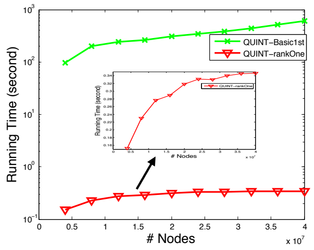


Figure 10: Running time per query vs. number of nodes on LiveJournal dataset (y-axis is in log scale). Inset: scalability of QUINT-rankOne only (y-axis is in linear scale).

Figure 11: Running time per query vs. number of edges on LiveJournal dataset (y-axis is in log scale). Inset: scalability of QUINT-rankOne only (y-axis is in linear scale).



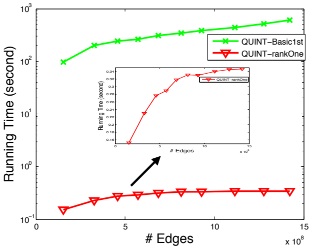


Figure 12: Running time per query vs. number of nodes on Twitter dataset (y-axis is in log scale). Inset: scalability of QUINT-rankOne only (y-axis is in linear scale).

Figure 13: Running time per query vs. number of edges on Twitter dataset (y-axis is in log scale). Inset: scalability of QUINT-rankOne only (y-axis is in linear scale).

[32], so that the random walks can be guided towards/away from certain specific zones in the network.

Second (Q2), an implicit assumption behind most existing works is that one common (optimal) network exists for all queries. In some scenarios, it is desirable to find a query-specific optimal network. Some recent work aim at finding k nearest neighbor nodes without calculating the ranking scores of all nodes for a given query node [7, 34, 39]. However, their focus is more on the computational efficiency.

Third (Q3), scalable computation is a key challenge for node proximity measures, and various solutions have been proposed to alleviate this issue. Lofgren et al. [21] propose to efficiently compute the RWR score between two given nodes under a certain error bound based on a bi-directional search. Yu and Lin [38] incrementally update the RWR scores when the underlying network structure changes without recomputing from scratch. Shin et al. [26] scale up RWR computation by reordering the adjacency matrix so that it contains a large and easy-to-invert submatrix. After the preprocessing, the RWR scores can be efficiently calculated. Zhang et al. [40] design a sampling method to accelerate the proximity computation based on the concept of random path. These efforts are complementary to the proposed algorithms in this paper - *after* we learn a query-specific optimal network, we can leverage these methods to speedup the subsequent proximity score computation.

Overall, none of the existing works simultaneously address all these challenges (i.e., Q1-Q3 summarized in Section 1).

Collaborative Filtering. Considering the problem setting, our work is also related to collaborative filtering [16, 14]. For example, we can apply collaborative filtering to learn the hidden relationships between nodes, and use the recovered weights as node proximity [36]; however, collaborative filtering still uses a fixed observed network/matrix as input. Ruchansky et al. [25] propose to use side information to complete a matrix/network (i.e., to infer user preference) by actively probing a subset of true underlying matrix; in contrast, our goal is to use the user preference to learn the optimal network for a given query.

Link Prediction. Finally, our work is also related to link prediction [19, 6]. While link prediction aims at predicting the existence of a network edge, our goal is to learn an optimal network where edges can be added, strengthened or weakened. RWR score computation based on our *learned* network could be used to improve link prediction. For example, we can directly use the RWR scores to predict links [20], or merge RWR scores with other inputs such as node attributes for link prediction [37, 12].

6. CONCLUSIONS

In this paper, we study the problem of learning query-specific optimal networks for node proximity measures. The proposed QUINT algorithm advances the state of the art in multiple dimensions, with a larger *optimization scope*, at a finer *optimization granularity*, and with a much better *optimization efficiency*. The extensive empirical evaluations on a diverse set of real networks show that the proposed algorithms (1) consistently outperform all the existing methods on all six commonly used metrics; (2) scale (sub)-linearly to billion-scale networks; (3) respond in a fraction of a second.

7. ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under Grant No. IIS1017415, by DTRA under the grant number HDTRA1-16-0017, by Army Research Office under the contract number W911NF-16-1-0168, by National Institutes of Health under the grant number R01LM011986, Region II University Transportation Center under the project number 49997-33 25 and a Baidu gift. Jie Tang is supported by the National 863 Program (2014AA015103, 2015AA124102) and NSFC (2014CB340506, 2012CB316006).

8. REFERENCES

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[2] A. Agarwal and S. Chakrabarti. Learning random walks to rank nodes in graphs. In *ICML*, 2007.

[3] A. Agarwal, S. Chakrabarti, and S. Aggarwal. Learning to rank networked entities. In *KDD*, pages 14–23. ACM, 2006.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[5] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, pages 635–644. ACM, 2011.

[6] N. Barbieri, F. Bonchi, and G. Manco. Who to follow and why: link prediction with explanations. In *KDD*, pages 1266–1275. ACM, 2014.

[7] P. Bogdanov and A. Singh. Accurate and scalable nearest neighbors in large networks based on effective importance. In *CIKM*, pages 1009–1018. ACM, 2013.

[8] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, 1998.

[9] Y.-C. Chen, Y.-S. Lin, Y.-C. Shen, and S.-D. Lin. A modified random walk framework for handling negative ratings and generating explanations. *ACM Transactions on Intelligent Systems and Technology*, 4(1):12, 2013.

[10] H. Cheng, P.-N. Tan, J. Sticklen, and W. F. Punch. Recommendation via query centered random walk on k-partite graph. In *ICDM*, pages 457–462. IEEE, 2007.

[11] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD*. ACM, 2012.

[12] N. Z. Gong, A. Talwalkar, L. Mackey, L. Huang, E. C. R. Shin, E. Stefanov, E. R. Shi, and D. Song. Joint link prediction and attribute inference using a social-attribute network. *ACM TIST*, 5(2):27, 2014.

[13] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272. IEEE, 2008.

[14] P. Jain, P. Netrapalli, and S. Sanghavi. Low-rank matrix completion using alternating minimization. In *STOC*, pages 665–674. ACM, 2013.

[15] M. Kim and J. Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *SDM*, pages 47–58, 2011.

[16] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, pages 426–434. ACM, 2008.

[17] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[18] L. Li, H. Tong, N. Cao, K. Ehrlich, Y.-R. Lin, and N. Buchler. Replacing the irreplaceable: Fast algorithms for team member recommendation. In *WWW*, pages 636–646, 2015.

[19] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. In *CIKM*, 2003.

[20] R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla. New perspectives and methods in link prediction. In *KDD*, pages 243–252. ACM, 2010.

[21] P. A. Lofgren, S. Banerjee, A. Goel, and C. Seshadhri. Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *KDD*, pages 1436–1445. ACM, 2014.

[22] J. Ni, H. Tong, W. Fan, and X. Zhang. Inside the atoms: ranking on a network of networks. In *KDD*, pages 1356–1365, 2014.

[23] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658. ACM, 2004.

[24] K. B. Petersen and M. S. Pedersen. The matrix cookbook, nov 2012.

[25] N. Ruchansky, M. Crovella, and E. Terzi. Matrix completion with queries. In *KDD*, pages 1025–1034. ACM, 2015.

[26] K. Shin, J. Jung, S. Lee, and U. Kang. Bear: Block elimination approach for random walk with restart on large graphs. In *SIGMOD*. ACM, 2015.

[27] D. Surian, N. Liu, D. Lo, H. Tong, E.-P. Lim, and C. Faloutsos. Recommending people in developers' collaboration network. In *WCRE*, 2011.

[28] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[29] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.

[30] H. Tong, C. Faloutsos, and J. Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622, 2006.

[31] H. Tong, B. Prakash, C. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. Chau. On the vulnerability of large graphs. In *ICDM*, pages 1091–1096, 2010.

[32] H. Tong, H. Qu, and H. Jamjoom. Measuring proximity on graphs with side information. In *ICDM*, pages 598–607, 2008.

[33] W. Wang and M. A. Carreira-Perpinán. Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application. *arXiv:1309.1541*, 2013.

[34] Y. Wu, R. Jin, and X. Zhang. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *SIGMOD*, pages 1139–1150. ACM, 2014.

[35] L. Yan, R. Dodier, M. Mozer, and R. Wolniewicz. Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. *ICML*, pages 848–855, 2003.

[36] Y. Yao, H. Tong, G. Yan, F. Xu, X. Zhang, B. K. Szymanski, and J. Lu. Dual-regularized one-class collaborative filtering. In *CIKM*. ACM, 2014.

[37] Z. Yin, M. Gupta, T. Weninger, and J. Han. A unified framework for link recommendation using random walks. In *ASONAM*, pages 152–159. IEEE, 2010.

[38] W. Yu and X. Lin. Irwr: Incremental random walk with restart. In *SIGIR*, pages 1017–1020. ACM, 2013.

[39] C. Zhang, S. Jiang, Y. Chen, Y. Sun, and J. Han. Fast inbound top-k query for random walk with restart. In *PKDD*, pages 608–624, 2015.

[40] J. Zhang, J. Tang, C. Ma, H. Tong, Y. Jing, and J. Li. Panther: Fast top-k similarity search on large networks. In *KDD*, 2015.

[41] L. Zheng, C. Shen, L. Tang, T. Li, S. Luis, and S.-C. Chen. Applying data mining techniques to address disaster information management challenges on mobile devices. In *KDD*, pages 283–291, 2011.

[42] D. Zhou, B. Schölkopf, and T. Hofmann. Semi-supervised learning on directed graphs. In *NIPS*, 2004.