Optimizing Knowledge Graphs through User Votes

Abstract—**Knowledge graphs have been widely used in many application domains. For example, in question answering (Q&A) systems, given a new question, we may use a knowledge graph to automatically identify the most suitable answers based on similarity evaluation. However, such systems may suffer from two major limitations. First, the knowledge graph constructed based on source data may contain errors. Second, the knowledge graph may become out-of-date and cannot quickly adapt to new knowledge. To address these issues, in this paper, we propose an interactive framework that refines and optimizes knowledge graphs through user votes. We develop an efficient similarity evaluation notion, called *extended inverse P-distance*, based on which the graph optimization problem can be formulated as a signomial geometric programming problem. We then propose a basic single-vote solution and a more advanced multiple-votes solution for graph optimization. We also propose a split-and-merge optimization strategy to scale up the multiple-votes solution. Extensive experiments based on real-life and synthetic graphs demonstrate the effectiveness and efficiency of our proposed framework.**

I. Introduction

Knowledge graphs have been widely used in a variety of applications, such as question answering (Q&A) systems [1], recommender systems [2][3], Web search engines [4], and precision medicine [5]. For example, measuring the similarity between questions and HELP documents using a knowledge graph has been shown to be effective in finding the most accurate answer [6].

In a knowledge graph, edges are essential to capture the relationship of two nodes and the strength of a relationship is typically represented by a weight of the edge. Clearly, how to assign edge weights is a key challenge for the construction and maintenance of a knowledge graph. Existing methods are mostly based on direct inference of the relationships between two objects (e.g., the hyperlinks between web pages) [7] or statistical information [6]. However, such methods are often vulnerable to source data errors or statistical errors [8]. Even worse, the relationships may become out-of-date as the knowledge evolves over time. For example, Prof. *Michael Jordan* is well known for his research on *statistical machine learning*. However, he has recently developed a new research interest in *computational biology*. As such, the weight of the edge <*Michael Jordan*, *statistical machine learning*> should decline, while that of the edge <*Michael Jordan, computational biology*> should rise.

To address the aforementioned issues, inspired by Human-In-The-Loop (HITL) model and Back Propagation (BP) [9], in this paper we propose an interactive framework that refines and optimizes edge weights of knowledge graphs through user votes. As shown in Fig. 1, upon receiving the user's question, the Q&A system returns a ranked list of documents $D = \langle d_1, d_2, d_3 \rangle$ based on similarity evaluation via a knowledge graph. After that, the user can check the answers and may vote for the best answer. For example, if the user finds the second document d_2 is most helpful to the question, he/she can vote this document for the best answer, suggesting that d_2, instead of d_1, should be ranked at the top. With such user feedback, we may update the edge weights (e.g., the weight of edge a is changed from 0.6 to 0.45 and that of edge b is changed from 0.4 to 0.55) so that d_2 will be ranked higher next time if a similar question is asked.

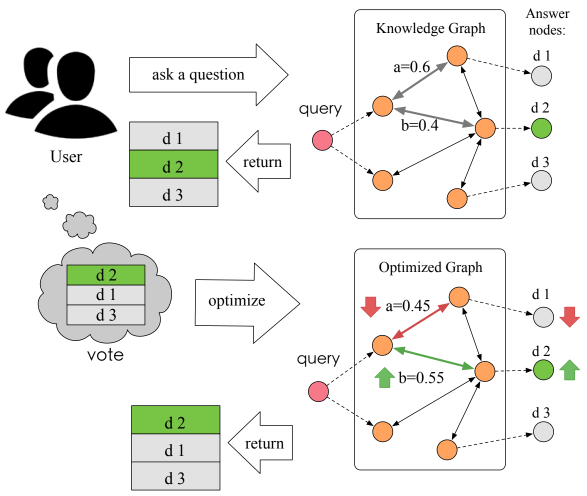


Fig. 1. An example of optimizing the knowledge graph based on user votes

We remark that voting mechanisms have been commonly employed by online systems to improve user experience [10]. Many Q&A websites, such as Quora, Yahoo! Answers, and Zhihu, use upvotes and downvotes to sort the responses posted by users. Nevertheless, these systems simply count the votes as a measure of trustworthiness for user-posted responses, and none of them has incorporated automated knowledge graph techniques for answering new questions. In contrast, this paper proposes to leverage user voting to optimize knowledge graphs that can be used to derive ranked answers for new questions. Furthermore, the proposed framework is not limited to Q&A systems. It can be extended to other knowledge graph-based applications:

Example 1: *An e-commerce company recommends commodities for customers based on similarity correlations of a product knowledge graph. If the company finds that customers are more likely to purchase the commodity in the recommendation list that does not rank first, it may want to optimize the* *graph by using implicit user voting information (i.e., purchase behaviors regarding recommended commodities).*

Example 2: *A search engine returns relevant pages given a user's query based on similarity evaluation. The click events, which indicate users' choices for the search results, are indeed implicit user votes. Our framework can make the search engine achieve higher accuracy by optimizing the underlying knowledge graph.*

There are a number of challenges in optimizing knowledge graphs through user votes. First, a knowledge graph is usually at large scale with complex structures, which entails update of a large number of edges in response to a user vote. Second, edge weights are continuous numerical values such that the search space of the updates is infinite. Third, it is inefficient to handle a large amount of votes simultaneously in terms of both computational time and space. Our framework addresses these challenges by analyzing the votes in an efficient manner. Specifically, to address the first and second challenges, we propose a new notion called *extended inverse P-distance*, which is a variant of personalized PageRank [11], to evaluate the similarity between the nodes in a knowledge graph. Based on this, we transform the graph optimization problem into a signomial geometric programming problem [12]. To address the last challenge, we develop a split-and-merge optimization strategy to divide the whole vote set into several sub-sets for processing, which reduces the time and space costs. In summary, our main contributions are as follows:

- We propose a novel framework for optimizing edge weights of a knowledge graph based on user votes. To the best of our knowledge, this is the first work that exploits user vote-based feedback to improve the quality of knowledge graphs.
- We develop an efficient similarity evaluation notion, based on which the graph optimization problem can be transformed into a signomial geometric programming problem.
- We develop a basic single-vote solution and a more advanced multiple-votes solution for graph optimization. We also propose a split-and-merge strategy to optimize the multiple-votes solution.
- We conduct extensive experiments on real-life and synthetic graphs. Experiments show that our framework significantly improves the accuracy of the knowledge graph in question answering. Moreover, we demonstrate that our optimization strategy can achieve more than 6X speedups compared to the baseline algorithm when handling the multiple-votes problem.

The rest of this paper is organized as follows. We first review related work in Section II. We present some preliminary knowledge in Section III. Next, we describe the basic single-vote method in Section IV and discuss the multiple-votes solution in Section V. In Section VI, we focus on how to accelerate the processing for large-scale votes. Section VII reports experimental results in terms of effectiveness and efficiency. Finally, we conclude our work in Section VIII.

II. Related Work

This section presents the related work on data cleaning and graph similarity measurement.

Data Cleaning. Data cleaning (or data cleansing) is a well-studied technique that aims to correct or remove inaccurate data from a database. Many data cleaning methods have been proposed in the past. Wang *et al.* [13] propose an approach that cleans sampled data to reduce the bias of queries with confidence intervals. Geerts *et al.* [14] introduce a uniform framework to resolve the repairing problem that involves different kinds of constraints and strategies. Bergman *et al.* [15] and Assadi *et al.* [16] employ experts with an optimized method to prune the errors in query results. Several existing works, such as [17] [18], leverage human's feedback to repair the errors and hence improve the quality of data. Our work is inspired by these works to integrate user feedback for graph data cleaning.

A knowledge graph is an effective way to represent the knowledge of the objective world [6]. Recently, Yang *et al.* [6] propose an efficient method to answer technical questions based on a knowledge graph. Some works have been proposed to clean the knowledge graph. A typical approach is based on coupling constraints [19]. For example, <*Michael Bloomberg*, *isMayorOf*, *NewYork*> implies that <*NewYork*, *isA*, *City*>. Liang *et al.* [20] derive new *isA* relationships according to the transitivity of the *isA* relation in the knowledge graph. Liang *et al.* [21] propose to supplement the *isA* relationship by using the collaborative filtering method. Lin *et al.* [22] suggest cleaning a probabilistic graph for reachability queries by crowdsourcing. However, to the best of our knowledge, no existing work has studied how to automatically optimize a knowledge graph based on user votes.

Graph Similarity Measurement. Measurement of similarity between nodes in a graph is important to many applications, such as social network analysis, information retrieval, and knowledge-graph-based question answering. A number of models have been proposed to capture the similarity of two nodes, such as Random Walk With Restart (RWR) [23], Personalized PageRank (PPR) [11], and SimRank (SR) [24]. These models can be classified into two categories according to the perspective of the edges between nodes. The first is based on an intuition that two objects are *similar* if they are referenced by *similar* objects, e.g., SimRank. The other is to regard the probability of walking from one node to another as the similarity between the nodes, e.g., RWR and PPR.

III. Preliminary and Problem Definition

In this section, we first present some preliminaries and then introduce the problem definition of our work.

A. Preliminary

Personalized PageRank. Personalized PageRank (PPR) [11], a variant of PageRank, is commonly used to measure the similarity between a query and the possible answers in a knowledge graph. Formally, we denote a directed graph by $G = (V, E, W)$, where V is the set of the nodes,

E is the set of edges, and W is the set of edge weights. For each node $v \in V$, the out-neighborhood and in-neighborhood of v are denoted by $O(v)$ and $I(v)$, respectively. The weight of a directed edge connecting node i to node j is denoted by $w(i,j)$, and $w(i,j) = \frac{1}{|O(i)|}$. An adjacency matrix is denoted by $\boldsymbol{A}$, where $\boldsymbol{A}_{ij} = w(j,i)$ if node j is connected to node i, and $\boldsymbol{A}_{ij} = 0$ otherwise. For ease of presentation, the nodes in the knowledge graph are numbered from 1 to $|V|$, where $|V|$ is the number of nodes in the graph. According to the definition of PPR in [25], given a target node v_q, the personalized PageRank vector $\boldsymbol{\pi}_{v_q}$ is the stationary distribution of the following random walk starting from v_q: at each step, either return to v_q with a probability of c, or move to a random out-neighbor of the current node otherwise. Each personalized PageRank vector $\boldsymbol{\pi}_{v_q}$ is of length $|V|$, where $\pi_{v_q}(v)$ denotes the v-th component of $\boldsymbol{\pi}_{v_q}$. In addition, $\boldsymbol{u}_{v_q}$ is a *preference vector*, where $u_{v_q}(v)$ denotes the amount of preference for node v, and $\sum_1^{|V|} u_{v_q}(v) = 1$. Since we want to measure the similarity between the target node v_q and other nodes, the target node v_q is the only preference node. Therefore, we set $u_{v_q}(v) = 1$ if $v = v_q$ and $u_{v_q}(v) = 0$ if $v \neq v_q$.

For a given $\boldsymbol{u}_{v_q}$, the personalized PageRank equation can be written as

$$\boldsymbol{\pi}_{v_q} = (1-c)\boldsymbol{A}\boldsymbol{\pi}_{\boldsymbol{v_q}} + c\boldsymbol{u}_{v_q} \tag{1}$$

where typically $c \approx 0.15$, and experiments have shown that small changes in c have little effect in practice [11]. A solution $\boldsymbol{\pi}_{v_q}$ to Equation (1) is a steady-state distribution of similarity measures between node v_q and other nodes.

Signomial Geometric Programming. Inspired by [17], our framework models the weighted graph optimization problem as a signomial geometric programming (SGP) problem, which is a type of nonlinear constrained optimization. According to the description of SGP in [12], the form of SGP is shown as follows:

$$SGP(X)\begin{cases} minimize & f_0(x) \\ s.t. & f_i(x) \leq 1, i = 1, \cdots, m, \\ & X = \{x : 0 < x^l \leq x \leq x^u\} \end{cases} \tag{2}$$

where

$$f_i(x) = \sum_{k=1}^{K_i} c_{i_k} x_1^{a_{i_{1k}}} x_2^{a_{i_{2k}}} \cdots x_n^{a_{i_{nk}}}, i = 0, \cdots, m \tag{3}$$

Here coefficients $c_{i_k} \in \boldsymbol{R}$ and $a_{i_{jk}} \in \boldsymbol{R}$, x^l and x^u are the lower bound and upper bound of x, respectively, and K_i is a positive integer number. A function of the form $f(x)$ is called *signomial function*. $X = \langle x_1, \cdots, x_n\rangle$ is a set of undetermined variables. $f_i(x) \leq 1$ $(i = 1, \cdots, m)$ is a series of constraint functions and seeking the minimum of $f_0(x)$ is the objective of the programming.

B. Problem Formulation

Graph and Node Similarity. Let $G = (V, E, W)$ denote a directed weighted graph where V and E are the sets of vertices and edges, respectively, and $w(i,j) \in W$ is the weight of the directed edge connecting node i to node j. The set of query nodes is denoted by $Q = \langle v_{q_1}, v_{q_2}, \cdots, v_{q_n}\rangle$ and the set of answer nodes is denoted by $A = \langle v_{a_1}, v_{a_2}, \cdots, v_{a_n}\rangle$. Note that Q and A are linked to the knowledge graph G, but $Q \cap V = \emptyset$ and $A \cap V = \emptyset$.

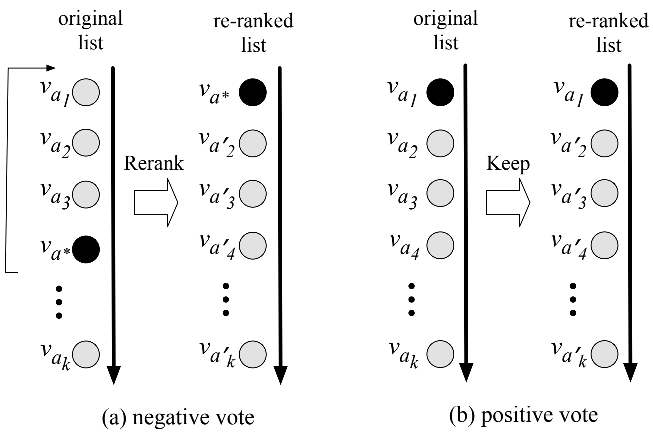


Fig. 2. An example of negative vote and positive vote

Definition 1: **Query-Answer Similarity Measure.** Given a query node v_q and an answer node v_a, the similarity between them is denoted as $S(v_q, v_a)$. Define an adjacency matrix $\boldsymbol{A}$, where $\boldsymbol{A}_{ij} = w(j,i)$ if node j is connected to node i and $\boldsymbol{A}_{ij} = 0$ otherwise. Following the definition of personalized PageRank, given a preference vector $\boldsymbol{u}_{v_q}$, $S(v_q, v_a)$ is defined as follows:

$$S(v_q, v_a) = \pi_{v_q}(v_a) \tag{4}$$

where $\pi_{v_q}(v_a)$ is the entry v_a of the solution $\boldsymbol{\pi}_{v_q}$ to Equation (1).

For each query node $v_q \in Q$, we assume that our framework returns a ranked list of top-k candidate answers $A_k = \langle v_{a_1}, v_{a_2}, \cdots, v_{a_k}\rangle$, which is sorted by the similarity score $S(v_q, v_a)$, and $A_k \subset A$.

Negative Votes and Positive Votes. The answer that is considered the best among all returned answers is termed as *the best answer*. A negative vote picks the answer that does not rank first in the returned answer list for the best answer. In contrast, a *positive vote* confirms that the first answer in the returned list is the best one. More formally, we define negative votes and positive votes as follows:

Definition 2: **Top Element.** Given a ranked list $E_k = \langle e_1, \cdots, e_k\rangle$, the first element of the list is denoted as $F_{top}(E_k)$. Formally, $F_{top}(E_k)$ is defined as:

$$F_{top}(E_k) = e_1 \tag{5}$$

Definition 3: **Negative Vote.** Given a ranked list $A_k = \langle v_{a_1}, \cdots, v_{a^*}, \cdots, v_{a_k}\rangle$ of candidate answers and the best answer v_{a^*} which does not rank first in the list A_k, the negative vote t_n is a mapping between two ranked lists:

$$t_n : A_k \mapsto A_k^* \tag{6}$$

such that $F_{top}(A_k^*) = a^*$. Furthermore, we denote the best answer v_{a^*} in the negative vote t_n as $v_{a^*_{t_n}}$. An example of the negative vote is shown in Fig. 2(a). The best answer v_{a^*} ranks fourth in the original list but rises to the first in the re-ranked list. Note that we do not consider the rankings of other answers.

We use T_n to denote the set of negative votes $T_n = \{t_n^1, \cdots, t_n^m\}$ and we call it the *negative vote set*.

Definition 4: **Positive Vote.** Given a ranked list $A_k = \langle v_{a_1}, \cdots, v_{a_k} \rangle$, where v_{a_1} is considered the best answer, the positive vote t_p is a mapping between two ranked lists:

$$t_p : A_k \mapsto A_k' \tag{7}$$

such that $F_{top}(A_k') = v_{a_1}$. We denote the best answer v_{a_1} in the positive vote t_p as $v_{a^*_{t_p}}$. As shown in Fig. 2(b), the best answer v_{a_1} keeps ranking first in A_k'.

We use T_p to denote the set of positive votes $T_p = \{t_p^1, \cdots, t_p^m\}$ and we call it the *positive vote set*.

Problem Objective. The objective of our framework is to adjust the edge weights in the knowledge graph according to the negative and positive votes submitted by users. Let $G^* = (V, E, W^*)$ denote the knowledge graph with the adjusted edge weights. Considering the potential conflicts between votes in practical applications, the objective of our problem is formalized as follows:

Definition 5: **Optimization Objective.** Given a set of negative votes T_n and a set of positive vote T_p, the score of the graph G^* is defined as:

$$\Omega(G^*) = \sum_{t \in T_n \cup T_p} (rank_t - rank_t'). \tag{8}$$

where t is a user vote in T_n and T_p; $rank_t$ is the position of $v_{a_t^*}$ in the original answer list, e.g., if $v_{a_t^*}$ ranks second in the original list, $rank_t = 2$; Similarly, $rank_t'$ is the position of $v_{a_t^*}$ in the re-ranked answer list, e.g., if $v_{a_t^*}$ ranks first in the re-rank list, $rank_t' = 1$. Based on this definition, the optimization objective is to find a G^* such that

$$G^* = \arg\max(\Omega(G^*)) \tag{9}$$

More intuitively, we seek the maximum increase of the ranking of the best answer for all negative votes $t_n \in T_n$ and the minimum decrease of the ranking of the best answer for all positive votes $t_p \in T_p$. By doing so, we hope that our framework will optimize the knowledge graph so that higher accuracy can be achieved for future queries.

IV. Single-Vote Solution

In this section, we propose a basic *single-vote* approach by considering negative votes individually. The main idea of this approach is transforming the graph optimization problem into an SGP problem by encoding the user votes as constraint functions. We first propose an equivalence equation of PPR to quickly evaluate the similarity between questions and answers and return the top-k answers for a question. We then describe the process of encoding that transforms a negative vote into a set of constraint functions and define the objective function of SGP. Finally, we present the complete procedure of the single-vote solution.

A. Similarity Evaluation

To explain the proposed approach, we use a running example of knowledge-graph-based question answering. Recall from Section III-A that we employ the PPR to measure the similarity between questions and answers. However, computing $\boldsymbol{\pi}_{v_q}$ naively using a fixed-point iteration requires multiple scans of the graph [11], which would incur prohibitively high time complexity. Furthermore, the solution of Equation (1) presents the similarity scores between the target node and all other nodes in the graph, which is not necessary for our question answering system.

Inspired by [24][25], we propose an *extended inverse P-distance*, a notion based on *inverse P-distance* that was introduced in [25]. Extended inverse P-distances are equivalent to the personalized PageRank scores, which can evaluate the similarity between the target node and any other selected node. Formally, we define the extended inverse P-distance $\Phi(v_q, v_a)$ from v_q to v_a as:

$$\Phi(v_q, v_a) = \sum_{z : v_q \rightsquigarrow v_a} P[z] c (1-c)^{|z|} \tag{10}$$

where the summation is taken over all paths starting at v_q and ending at v_a, possibly touching some nodes in the graph multiple times. For a path $z = \langle v_q, v_1, \cdots, v_k, v_a \rangle$, the length $|z|$ is $k+1$. The probability of z, denoted by $P[z]$, is defined as:

$$P[z] = w(v_q, v_1) w(v_k, v_a) \prod_{i=1}^{k-1} w(v_i, v_{i+1}), i = 1, 2, \cdots \tag{11}$$

where $w(v_i, v_{i+1})$ represents the edge weight from node v_i to node v_{i+1}. For a special case in which $|z| = 2$, $P[z] = w(v_q, v_1) w(v_1, v_a)$. If there is no path from v_q to v_a, the extended inverse P-distance $\Phi(v_q, v_a) = 0$. As proven in Theorem 1, the personalized PageRank scores can be represented by the extended Inverse P-distances in a weighted graph.

Theorem 1: The PPR vector scores can be represented by the extended inverse P-distances in a weighted graph.

Proof (Sketch): We prove their equivalence in two steps. First, we develop a decomposition theorem to compute each dimension of the PPR vector, which is an equivalent way of formalizing the personalized PageRank scores. Second, we prove that the extended inverse P-distance is equivalent to the PPR score derived from the decomposition theorem. See the Appendix for detailed proof. □

Based on Theorem 1, the similarity between question v_q and answer v_a can be evaluated by the extensive inverse P-distance as follows:

$$S(v_q, v_a) = \Phi(v_q, v_a) \tag{12}$$

Example: An example of knowledge graph is shown in Fig. 3. In order to compute $S(v_q, v_{a_3})$, we first search all paths from question v_q to answer v_{a_3}: $\langle v_q \rightharpoonup v_8 \rightharpoonup v_{10} \rightharpoonup v_{a_3}; v_q \rightharpoonup$

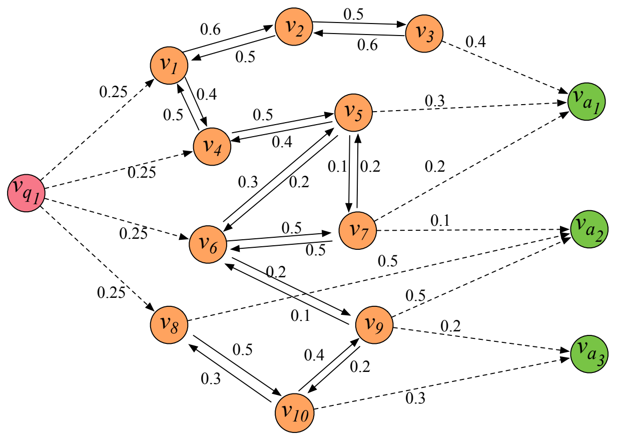


Fig. 3. An example of similarity evaluation based on the knowledge graph

$v_6 \rightharpoonup v_9 \rightharpoonup v_{a_3}; v_q \rightharpoonup v_8 \rightharpoonup v_{10} \rightharpoonup v_9 \rightharpoonup v_{a_3}; v_q \rightharpoonup v_8 \rightharpoonup v_{10} \rightharpoonup v_9 \rightharpoonup v_{10} \rightharpoonup v_{a_3}; \cdots\rangle$. We then use Equation (12) to compute the similarity $S(v_q, v_{a_3})$ as follows:

$$\begin{aligned} S(v_q, v_{a_3}) =& (0.25 * 0.5 * 0.3) * c * (1-c)^3 \\ &+ (0.25 * 0.5 * 0.4 * 0.2) * c * (1-c)^4 \\ &+ (0.25 * 0.2 * 0.2) * c * (1-c)^3 \\ &+ (0.25 * 0.5 * 0.4 * 0.2 * 0.3) * c * (1-c)^5 \\ &+ \cdots \end{aligned}$$

Given a question v_q, we compute the similarity $S(v_q, v_a)$ with each possible answer and return a ranked list of top-k answers to the user. To compute $S(v_q, v_a)$, in theory, we need to identify all paths from node v_q to node v_a, which is time consuming. Since each edge weight $w(v_i, v_{i+1})$ is less than 1, the probability $P[z]$ degrades exponentially as the length of the path increases. Thus, to speed up the similarity computation, we prune the path with a length longer than L. More details on the setting of K will be discussed in Section VII-D.

B. Encoding Negative Votes

Under the condition of solving single votes, we do not consider positive votes, since in those cases the best answer has been ranked first in the ranked list. Given a negative vote $t_n \in T_n$, we encode it as a set of constraint functions and hence the graph optimization problem is transformed into an SGP programming problem. As mentioned above, we employ the extended inverse P-distance, which is a signomial function, to evaluate the similarity between questions and answers.

An SGP problem consists of two parts: constraint functions and objective function. In the following, we describe the details of constructing the SGP problem for each of these two parts.

Constraint Functions. To encode a negative vote, we first introduce a real-valued variable $x_{i,j}$ to represent the edge weight from node i to node j. The initial value of $x_{i,j}$ equals the edge weight $w(i,j)$ in the knowledge graph. Then, we analyze the user negative vote. Recall Definition 3 from Section III-B that a negative vote specifies the best answer v_{a^*} suggested by the user. Hence, the similarity of question v_q to the best answer v_{a^*} should be larger than that to other answers v_a in the list. Thus, we define the constraint functions as follows:

$$s.t. \begin{cases} S(v_q, v_{a^*}) > S(v_q, v_{a_1}) \\ S(v_q, v_{a^*}) > S(v_q, v_{a_2}) \\ \qquad \cdots \\ S(v_q, v_{a^*}) > S(v_q, v_{a_{k-1}}) \end{cases} \tag{13}$$

More formally, we substitute Equation (12) for $S(v_q, v_a)$ and rewrite the inequalities in the standard format of SGP:

$$s.t. \begin{cases} \sum_{z:v_q \rightsquigarrow v_{a_1}} P[z]c(1-c)^{|z|} - \sum_{z:v_q \rightsquigarrow v_{a^*}} P[z]c(1-c)^{|z|} < 0 \\ \qquad \cdots \\ \sum_{z:v_q \rightsquigarrow v_{a_{k-1}}} P[z]c(1-c)^{|z|} - \sum_{z:v_q \rightsquigarrow v_{a^*}} P[z]c(1-c)^{|z|} < 0 \end{cases} \tag{14}$$

By satisfying the constraint functions, the problem maximizes the ranking increase of the best answer for the negative vote t_n, thereby maximizing the objective of our graph optimization problem specified in Definition 5. Since there could be many ways of updating the edge weights to achieve this, we set an objective function for the SGP problem.

Objective Function. Different objective functions can be chosen for the SGP problem in our framework. As with [17], we seek to minimize the magnitude of changes of edge weights, which is measured by the Euclidian distance $d(X, X^*)$ of edge weights between the initial variable set X and the optimized variable set X^*. More formally, denoting by $x_{i,j}$ and $x^*_{i,j}$ the variable of the edge weight from node i to node j in X and X^*, respectively, $d(X, X^*)$ is defined as follows:

$$d(X, X^*) = \sum_{x_{i,j} \in X, x^*_{i,j} \in X^*} (x^*_{i,j} - x_{i,j})^2 \tag{15}$$

C. Complete Procedure of Single-Vote Solution

We proceed to describe the complete procedure of the single-vote solution. The algorithm takes as input the negative vote set T_n, the edge weight variable set X and the initial weighted directed graph G, and outputs a new knowledge graph G^*.

As shown in Algorithm 1, for each $t_n \in T_n$, we first initialize the variable set X by using $ObtainVariableSet$ based on the corresponding edges in the current graph (Lines 3-8). Then, the function $GenerateConstraints$ encodes a single negative vote t_n as constraint functions sgl_cons (Line 9). After that, $GenerateObjective$ augments the program with an objective function that models the changes of edge weights between the current graph and the optimized graph (Line 10). The function $SGPsolver$ solves the SGP problem and generates an adjusted variable set X' which is used to update the weights of the corresponding edges in the

Algorithm 1 *Basic*: The Single-Vote Solution

```
input: $T_n, G$
output: $G^*$
1: $G^* \leftarrow G$
2: for each $t_n \in T_n$ do
3:     // obtain the variable set $X$
4:     $X \leftarrow ObtainVariableSet(t_n, G^*)$
5:     for each $x_{i,j} \in X$ do
6:         // Initialize the $x_{i,j}$ based on edge weights in $G^*$
7:         $x_{i,j} \leftarrow G^*_{i,j}$
8:     end for
9:     $sgl\_cons \leftarrow GenerateConstraints(t_n)$
10:    $sgl\_obj \leftarrow GenerateObjective(sgl\_cons, X)$
11:    $X' \leftarrow SGPsolver(sgl\_cons, sgl\_obj)$
12:    // update the edge weights in $G^*$
13:    for each $x'_{i,j} \in X'$ do
14:        $G^*_{i,j} \leftarrow x'_{i,j}$
15:    end for
16:    $G^* \leftarrow NormalizeEdges(G^*)$
17: end for
18: Return $G^*$
```

graph (Lines 11-15). Finally, $NormalizeEdges$ normalizes the edge weights (Line 16). After traversing each $t_n \in T_n$, the procedure returns an optimized graph G^*.

V. Multiple-Votes Solution

In this section, we propose a multiple-votes solution that processes user votes simultaneously, including a positive vote encoding strategy.

As discussed above, the single-vote solution encodes each negative vote $t_n \in T_n$ as a set of constraint functions of the SGP problem. However, in practical applications, there might be conflicts among user votes. Even worse, errors may occur in some user votes. A major drawback of our basic solution is that it cannot handle these problems, since it constructs the SGP problem and adjusts the edge weights for each negative vote individually. Due to the order of processing, the edge weight in the graph will be biased towards the last programming result, which probably reduces the overall quality of graph optimization. For example, if most of the programming results for negative votes increase the weight of an edge e, but the programming result of the last vote, which may be a low credible vote, decreases the weight of e, the final weight will be decreased.

To address this drawback, we propose a *multiple-votes solution* that processes all negative votes and positive votes in one batch. The benefit is two-folded. First, a positive vote represents a positive feedback for the current knowledge graph, which is now reflected in the solution. Second, since the constraint functions are encoded by multiple votes, the solver can automatically handle the conflicts among the votes in the process of solving the SGP problem.

Encoding Positive Votes. Recall the definition of positive vote (Definition 4), in which the best answer keeps ranking first in the re-ranked list. The essence of a positive vote represents a confirmation, which should also be considered in graph optimization. That is, a positive vote can be used to keep the best answer in the optimized graph.

Similar to negative votes, given a positive vote $t_p \in T_p$, we encode it as a set of constraint functions. We also introduce a real valued variable set X, which consists of variables $x_{i,j}$. The definition of $x_{i,j}$ is the same as that defined in Section IV-B, which represents the corresponding edge weight in the graph. A positive vote contains a re-ranked list A'_k of answers, where the best answer v_{a_1} keeps ranking first in the list. Hence, we define the constraint functions as follows:

$$s.t. \begin{cases} \sum\limits_{z:v_q \rightsquigarrow v_{a_2}} P[z]c(1-c)^{|z|} - \sum\limits_{z:v_q \rightsquigarrow v_{a_1}} P[z]c(1-c)^{|z|} < 0 \\ \cdots \\ \sum\limits_{z:v_q \rightsquigarrow v_{a_{k-1}}} P[z]c(1-c)^{|z|} - \sum\limits_{z:v_q \rightsquigarrow v_{a_1}} P[z]c(1-c)^{|z|} < 0 \end{cases} \tag{16}$$

Note that these constraint functions are similar to the constraint functions (Equation (14)) generated for a negative vote.

Multiple-Votes Solution. Similar to the single-vote solution, we formulate an SGP problem for the multiple-votes solution. Here, the SGP objective function is set the same as that in the single-vote solution. On the other hand, we combine the constraint functions for all negative votes and positive votes into a *comprehensive* set of constraint functions for the multiple-votes SGP problem. Specifically, for each negative vote $t_n \in T_n$ and each positive vote $t_p \in T_p$, we denote the questions in t_n and t_p as $v_{q_{t_n}}$ and $v_{q_{t_p}}$, respectively. The comprehensive set of constraint functions are defined as follows (for clarity, we use the original form to represent the constraints):

$$\begin{gathered} \forall t_n \in T_n, \forall v_{a_n} \in \{A(v_{q_{t_n}}) \backslash a^*_{v_{q_{t_n}}}\} \\ \forall t_p \in T_p, \forall v_{a_p} \in \{A(v_{q_{t_p}}) \backslash a_{1 q_{t_p}}\} \\ s.t. \begin{cases} S(v_{q_{t_n}}, a^*_{v_{q_{t_n}}}) > S(v_{q_{t_n}}, v_{a_n}) \\ S(v_{q_{t_p}}, a_{1 v_{q_{t_p}}}) > S(v_{q_{t_p}}, v_{a_p}) \end{cases} \end{gathered} \tag{17}$$

where $A(v_{q_{t_n}})$ and $A(v_{q_{t_p}})$ are the top-k answer set of $v_{q_{t_n}}$ and $v_{q_{t_p}}$ respectively; $a^*_{v_{q_{t_n}}}$ and $a_{1 q_{t_p}}$ are the best answer of $v_{q_{t_n}}$ and $v_{q_{t_p}}$, respectively.

If there are no conflicts among the user votes, it is likely that all constraint functions in Equation (17) can be satisfied. In this case, the multiple-votes problem can be easily solved by a normal SGP solver. However, there could be conflicts among the user votes. To deal with the case where not all constraint functions can be satisfied, we employ an Interior Point (IP) algorithm [26], which is widely used in constrained programming [27][28], to solve the multipe-votes SGP problem. The main idea of the IP algorithm is to construct a new objective function based on the constraint functions and the objective function in the original problem, which essentially transforms constrained programming to unconstrained programming. More specifically, the original constraint functions are modeled as a penalty in the new objective function to penalize constraint violations. Thus, the IP algorithm aims to satisfy as many constraints as possible. Theorem 2 proves that the IP algorithm maximizes the objective of our graph optimization problem (Definition 5).

Theorem 2: Solving the multiple-votes SGP problem by using the IP algorithm maximizes the optimization objective in Definition 5.

Proof: The proof is straightforward for the case where all constraint functions can be satisfied. We focus on the case where not all constraint functions can be satisfied. It is noted that each constraint function in the original multiple-votes SGP problem corresponds to the increase of one rank for negative votes or the decrease of one rank for positive votes. Since the IP algorithm aim to satisfy as many such constraint functions as possible, this is equivalent to maximizing the ranking increase of the best answer for negative votes and minimizing the ranking decrease of the best answer for positive votes. Thus, the optimization objective defined in Equation (8) is maximized. □

VI. Optimization for Multiple-Votes Solution

One issue with our multiple-votes solution is that a large amount of votes would lead to an exponential increase in solver time, due to the sharply increased number of variables and constraint functions. In this section, we propose a *split-and-merge* strategy to accelerate the processing of the multiple-votes solution.

A. Split and Merge

Since SGP is an NP-hard problem [29], we devise a split-and-merge strategy which is a heuristic algorithm to avoid the exponential increase in solver time for large-scale SGP problems. The main idea is to break the large problem into a set of small sub-problems, since it is faster to solve small problems. Furthermore, small problems can be solved in parallel by embracing distributed technologies. The rest of this subsection describes the two parts of the proposed split-and-merge strategy.

Splitting the Vote Set. Inspired by the approach of graph partition [30], we split a knowledge graph into several *clusters* based on the *similarity* between the votes. As mentioned in Section V, each negative or positive vote includes a question node and a set of top-k answer nodes. Recall that we prune the path with a length longer than L. Hence, all the edges associated with similarity evaluation of a vote are centrally distributed in a sub-graph of the graph. Thus, the large graph is split based on the edges associated with the votes. Intuitively, the votes with more common edges should be classified into the same cluster and the votes with fewer common edges should be separated, so that there are less conflicts between clusters.

Formally, we denote the set of edges associated with a vote $t \in T$ as $E(t)$. The similarity between two votes t_i and t_j is defined as follows:

$$Sim(t_i, t_j) = \frac{E(t_i) \cap E(t_j)}{E(t_i) \cup E(t_j)} \tag{18}$$

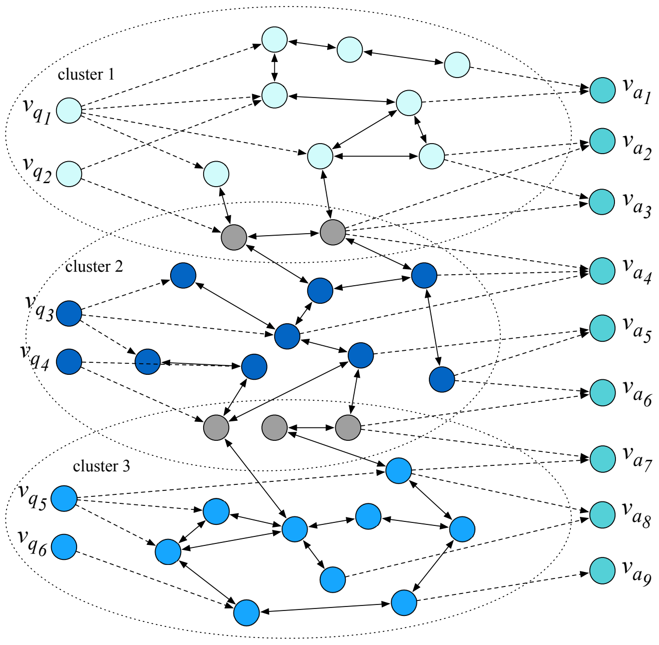


Fig. 4. An example of split strategy

We employ an affinity propagation (AP) clustering algorithm [31] to classify the votes based on $Sim(t_i, t_j)$ between the votes. The AP algorithm can automatically find the optimal number of clusters and make the number of common edges between neighboring clusters the minimum. Fig. 4 shows an exemplifying example, where a graph is divided into three clusters. After forming clusters, we construct an SGP problem for each cluster and use the multiple votes-solution to solve them separately.

We remark that the split strategy makes sense for real knowledge graphs. In the scenario like our running example, the entities (nodes) with high correlation centrally distributed in a sub-graph may represent a domain in the multi-domain knowledge graph. For example, the entities of athletes will be distributed in the sub-graph which represents *Sports*.

Merging the Results. After solving the SGP problems constructed for each cluster, we obtain a set of results. The results consisting of the change of each variable $x_{i,j} \in X$ will be merged into the knowledge graph. As mentioned above, the number of common edges between clusters is minimized by using the AP algorithm. In other words, most variables are changed in only one cluster. Therefore, the *merge strategy* focuses on the variables which are changed in multiple clusters. For each $x_{i,j} \in X$, the change of $x_{i,j}$ in the final result is denoted by $\Delta x_{i,j}$. The merge strategy is proposed as follows:

- If $x_{i,j}$ is changed in only one cluster, $\Delta x_{i,j}$ equals this change;
- If $x_{i,j}$ is changed in several clusters, $\Delta x_{i,j} = \#vote\langle \Delta x^1_{i,j}, \cdots, \Delta x^n_{i,j} \rangle$, where $\#vote\langle \Delta x^1_{i,j}, \cdots, \Delta x^n_{i,j} \rangle$ considers the changes in the related clusters by using a voting mechanism.

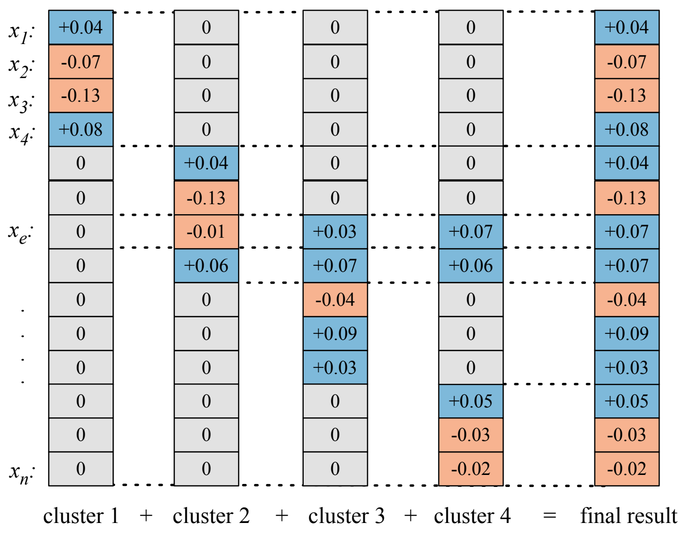


Fig. 5. An example of merge strategy

Specifically, the number of votes in a cluster $\mathcal{C}$ is denoted by $n_{\mathcal{C}}$. First, we determine the sign of $\Delta x_{i,j}$ by the sign of $\sum_{\mathcal{C}}(n_{\mathcal{C}} \cdot \Delta x^{\mathcal{C}}_{i,j})$. Then, we assign the maximum of $\langle \Delta x^1_{i,j}, \cdots, \Delta x^n_{i,j} \rangle$ to $\Delta x_{i,j}$ if the sign of $\Delta x_{i,j}$ is positive, otherwise the minimum is assigned to $\Delta x_{i,j}$.

Fig. 5 shows an example of four clusters. The change set of variable x_e is $\langle -0.01, +0.03, +0.07 \rangle$. We assume that the number of votes in each cluster is $n_2 = 10$, $n_3 = 8$, and $n_4 = 9$, respectively. Thus, the sign of $\Delta x_{i,j}$ is positive ($\sum_{\mathcal{C}} n_{\mathcal{C}} \Delta x^{\mathcal{C}}_{i,j} = 10*(-0.01)+8*0.03+9*0.07 \geq 0$). Therefore, we choose the maximum 0.07 of $\langle -0.01, 0.03, 0.07 \rangle$ as the final result. As can be seen, the merge strategy we employ is simple and efficient, while it tends to satisfy the results of most clusters by using the voting mechanism. The experimental results in Section VII-C confirm the effectiveness of the voting mechanism.

B. Complete Procedure

The complete procedure of graph optimization is shown in Algorithm 2. The algorithm takes as input the negative vote set T_n, the positive vote set T_p and the initial weighted directed graph G, and outputs a new knowledge graph G^*.

The function $AffinityPropagation$ classifies all votes into several clusters (Line 1). For each cluster, we construct an SGP problem. Specifically, $RetrieveT$ retrieves the corresponding vote set T' in $T\_cluster$ (Line 3). Next, we obtain the variable set X based on the original knowledge graph G by the function $ObtainVariableSet$ (Line 4) and initialize it by the function $InitializeVariables$ (Line 5). The process of constructing an SGP problem is similar to Algorithm 1 that uses $GenerateConstraints$ to encodes T' as a comprehensive set of constraint functions $multi\_cons$ (Line 6), uses $GenerateObjective$ to augment the program with an objective function (Line 7), and employs IP-based $SGPsolver$ to solve the problem (Line 8). Then, the result

Algorithm 2 *Optimization*: The Multiple-Votes Solution

```
input: $T_n, T_p, G$
output: $G^*$
1: $T\_cluster\_set \leftarrow AffinityPropagation(T_n, T_p)$
2: for each $T\_cluster \in T\_cluster\_set$ do
3:     $T' \leftarrow RetrieveT(T\_cluster)$
4:     $X \leftarrow ObtainVariableSet(T', G)$
5:     $X \leftarrow InitializeVariables(G)$
6:     $multi\_cons \leftarrow GenerateConstraints(T')$
7:     $multi\_obj \leftarrow GenerateObjective(multi\_cons, X)$
8:     $X_{part} \leftarrow SGPsolver(multi\_cons, multi\_obj)$
9:     $X_{set}.Add(X_{part})$
10: end for
11: $X' \leftarrow Merge(X_{set})$
12: $G' \leftarrow UpdateGraph(G, X')$
13: $G^* \leftarrow NormalizeEdges(G')$
14: Returen $G^*$
```

X_{part} is added into X_{set} (Line 9). After obtaining X_{set} which consists of the result of each cluster, $Merge$ merges them into the final result X' (Line 11). Finally, the edge weights are updated by $UpdateGraph$ based on X' and normalized by $NormalizeEdges$ (Lines 12-13).

VII. Performance Evaluation

In this section, we investigate the effectiveness and efficiency of our proposed framework. Our experiments are organized as follows. First, we study the effectiveness of our framework for a knowledge graph which is built based on the question answer pairs crawled from the *Taobao* customer service website.[1] We then evaluate the efficiency of our proposed solutions with several real-life graphs under controlled settings. We end with a study on the impacts of the path length and other parameter settings.

A. Experiment Setup

1) Datasets: Due to the difficulty of obtaining real user votes, our datasets are classified into two classes: a small real-life knowledge graph with real votes, and a large real-life graph with synthetically generated votes. The details of these datasets are given below.

Knowledge Graph with Real Votes. In order to investigate the effectiveness of our framework, we built a knowledge graph based on question-answer pairs. Specifically, we collected 2,379 questions together with their HELP documents from the online customer service of Taobao. We extracted the entities in the questions and documents, and built a knowledge graph with 1,663 nodes and 17,591 edges.

We recruited five volunteers to conduct a user study. We first let them to ask 100 questions. Each question is linked to some related entities (nodes) in the knowledge graph. For each question, we returned a ranked list of documents based on similarity evaluation in the knowledge graph. Then, we collected the user feedback in the form of votes, which

[1]https://www.taobao.com/

were divided into the *negative vote set* and *positive vote set*. There are 47 negative votes and 53 positive votes. Finally, we asked one domain expert to generate 100 questions and assigned the best HELP document for each question. These question-document pairs, serving as *test dataset*, are used for performance evaluation.

Knowledge Graph with Synthetic Votes. To investigate the efficiency of our framework, three real-life graphs were used in our experiments, which are available on *KONECT*.[2]

- **Twitter:** It is a directed network graph with 23,370 nodes and 33,101 edges. Each node represents a user on Twitter and each edge between two user nodes indicates their *follow* relationship.
- **Digg:** It is a directed network graph consisting of 30,398 nodes and 87,627 edges. This graph was built based on the reply information in a social news website Digg. Each node in the graph is a user of the website, and each directed edge denotes that a user replied to another user.
- **Gnutella:** This is a graph of Gnutella hosts since 2002. It has 62,586 nodes and 147,892 edges. Each node in the graph is a Gnutella host, and each directed edge indicates the connection between two hosts.

We generated a set of synthetic votes for each of these real-life graphs. Specifically, we generated N_Q queries and N_A answers randomly linked to a N_{nodes}-node subgraph, with an average degree N_{degree}. After evaluating the similarity between the queries and the answers, we obtained a ranked list of top-k answers for each query. Then, we generated a negative or positive vote by randomly selecting an answer in top-k answers as the best answer of the query. The average position of the best answers for negative votes is set at N_{aveN}. The default settings for these parameters are: $N_D = 1,000$, $N_{degree} = 4$, $N_{nodes} = 10,000$, $k = 20$, and $N_{aveN} = 10$.

In addition, we generated a series of *random* graphs to study the impacts of the graph parameters on the execution time. The statistics of all the graphs are described in Table I.

TABLE I
STATISTICS OF GRAPH DATASETS

DataSet	$\lvert V\rvert$	$\lvert E\rvert$	**Average Degree**
Taobao	1,663	17,591	10.57
Twitter	23,370	33,101	2.83
Digg	30,398	87,627	5.77
Gnutella	62,586	147,892	4.73
Random	5,000	-	-

2) Metrics: Effectiveness and efficiency are two main metrics to evaluate our proposed framework.

- **Effectiveness.** We use Ω_{avg} to measure the effectiveness of graph optimization, which corresponds to the optimization objective (Definition 5). Ω_{avg} is defined as follows:

$$\Omega_{avg} = \frac{\sum_{t \in T_n \cup T_p} (rank_t - rank'_t)}{|T_n| + |T_p|} \tag{19}$$

where the definitions of t, $rank_t$, and $rank'_t$ are the same as those in Definition 5. We also employ the MRR (Mean Reciprocal Rank) and MAP (Mean Average Precision), which are standard information retrieval measures, to study the effectiveness of our framework.
- **Efficiency.** The efficiency measures the elapsed time of solving the SGP problem in each algorithm. For comparison, we include the basic algorithm for the multiple-votes solution and the optimization algorithm to study the efficiency of our split-and-merge strategy. Also, we investigate Ω_{avg} of both algorithms to study their effectiveness in graph optimization. Besides, the scalability of our framework is measured by the number of user votes which can be solved in batch.

3) Experiment Environment: The experiments were conducted on the laptops (Intel Core i5 2.7 GHz CPU and 16GB RAM) running MATLAB as the SGP solver on Mac OS X 10.11.6 operating system. Each experiment was repeated 10 times. We report the average of the measured results.

B. Effectiveness of Graph Optimization

This first set of experiments examines the effectiveness of our frameworks for graph optimization. We use the knowledge graph *Taobao* with real user votes for this set of experiments. We evaluate the performance improvement of the optimized graph against the original knowledge graph in question answering. Specifically, we compare the single-vote solution that is only based on the negative votes, and the multiple-votes solution that is based on both the negative and positive votes. We use the test dataset to measure the effectiveness of the graph optimization.

TABLE II
RANKING OF BEST ANSWERS IN TEST DATASET

Graph	R_{avg}	Ω_{avg}	P_{avg}
Original Graph	3.56	-	-
Optimized by single-vote solution	3.59	-0.03	-0.8%
Optimized by multiple-votes solution	**2.86**	**0.67**	18.82%

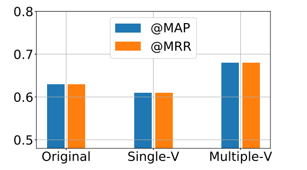
Denote the average ranking of the best answers and the average percentage-wise improvement of the rankings as R_{avg} and P_{avg}, respectively. We report the measurements of R_{avg}, Ω_{avg}, and P_{avg} in Table II. The average position of the best answers in the answer list is dropped from 3.56 to 3.59 and raised to 2.87, respectively. On average, the ranking of the best answers is degraded by 0.84% and promoted by 18.82% through the basic single-vote solution and the multiple-votes solution, respectively. Clearly, the multiple-votes solution is capable of improving the ranking of the best answers. On the other hand, the single-vote solution does not perform well. This is partly because that the single-vote solution optimizes the graph only based on the negative votes, which would degrade the ranking of the best answers that rank first in the original answer list. Another reason is that the single-vote solution cannot handle the conflicts among the votes, which also affects the performance of graph optimization.

[2]http://konect.uni-koblenz.de/

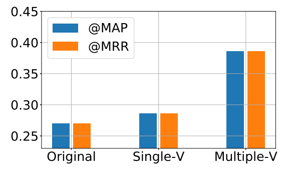
TABLE III
PROMOTION OF BEST ANSWERS IN TOP-K LIST

Method	H@1	H@3	H@5	H@10
IR	0.15	0.29	0.34	0.47
KG	0.49	0.69	0.79	0.90
KG (single-vote)	0.45	0.68	0.81	0.92
KG after (multi-votes)	**0.53**	**0.77**	**0.87**	**0.94**

Next, we study how much the graph optimization helps promote the best answers. We define H@k as the percentage of the questions in the test dataset whose best answers are ranked no lower than k. Besides the knowledge graph-based (KG) approach, we include an information retrieval-based (IR) approach for comparison. The IR approach compares the entities in the questions and documents and returns top-k answers based on their coincidence rates. The results are shown in Table III. All KG approaches significantly outperform the IR approach. Note that the results in top-1 and top-3 degrade after optimization by using the single-vote solution, due to the reasons mentioned in the last paragraph. Nevertheless, the results in top-5 and top-10 are improved, since the single-vote solution digs out the implicit information in user negative votes to adjust the edge weights. In all cases tested, the multiple-votes solution performs the best, which, on average, is 168%, 8.6%, and 10.1% better than the IR approach, the basic KG approach, and the KG (single-vote) approach, respectively.



(a) MAP and MRR for the whole test dataset



(b) MAP and MRR for part of test dataset

Fig. 6. MRR and MAP results of graph optimization for test dataset

Finally, Fig. 6 shows the MRR and MAP results. As shown in Fig. 6(a), the MRR and MAP degrade from 0.63 to 0.61 after optimization with the single-vote solution. In contrast, the multiple-votes solution that considers both positive and negative votes achieves about 8% improvement of answer ranks. To investigate the reason, we also show the results only for the questions whose best answers do not rank first in the original answer list. Both the single-vote solution and the multiple-votes solution achieve higher MRR and MAP scores. This suggests that the single-vote solution does help for promoting the non-top-1 answers. Its poor performance for the whole test dataset is mainly because it does not consider positive votes, which cannot prevent the top-1 answers from degrading after the graph optimization. This coincides with the results shown in Tables II and III.

C. *Efficiency and Effectiveness of Optimization Strategy*

The next set of experiments investigates the effectiveness of the split-and-merge optimization on the large real-life graphs with synthetic votes. In these experiments, we compare the elapsed time of the *basic* multiple-votes solution and the multiple-votes solution with the split-and-merge strategy by varying the number of votes. Moreover, we investigate Ω_{avg} of different solutions to study the impact of the split-and-merge strategy on graph optimization. The single-vote solution is also included for comparison.

As shown in Figs. 7(a)-(c), the elapsed time of the basic multiple-votes solution (Multiple-V) increases significantly as the number of votes grows. This is because the increase in the number of votes leads to an exponential increase in the number of variables and constraint functions in SGP. Hence, the solver time is dramatically increased. Note that the increase in the number of votes also leads to high memory consumption of the basic multiple-vote solution. Therefore, we cannot obtain the results of this solution when the space requirement exceeds the memory capacity (16GB) of our server (e.g., after the number of votes becomes larger than 70 in Fig. 7(a)).

In the split-and-merge (S-M) strategy, we select the median of the similarities between votes as the classification criterion, based on which the AP algorithm automatically classifies the votes into clusters. The average size of the classified clusters is 5 votes. Compared to the basic solution, the elapsed time of the solution with the S-M strategy is reduced significantly, by at least 6X when the number of votes grows beyond 70. Furthermore, it can be further optimized by using distributed technologies, since the clusters classified by the AP algorithm are independent of each other. We test a distributed approach with four computers to process the classified clusters. As can be seen, the distributed approach significantly improves the scalability by reducing the elapsed time by an order of magnitude.

In Figs. 7(d)-(f), we show the Ω_{avg} scores of different solutions. It is interesting to observe that the optimized multiple-votes solution is close to or even exceeds that of the basic solution. This implies that our optimization strategy can save a lot of computation time without sacrificing much on the performance of graph optimization.

Regarding the single-vote solution (Single-V), as shown in Figs. 7(a)-(c), it is faster than the multiple-votes solution with the split-and-merge strategy. Nevertheless, the effectiveness of the multiple-votes solution in graph optimization significantly outperforms the single-vote solution (Figs. 7(d)-(f)).

D. *Impact of Parameter Settings*

The following set of experiments is designed to justify the parameter choices. Specifically, we first study the impact of the path length on the similarity scores and the execution time. We then compare the results under different graph characteristics.

Path Pruning. Recall in the method of similarity evaluation in Section IV-A, we prune the paths with a length longer than L. In this experiment, we set $N_Q = 1$ query and return top-20 answers. We test the path pruning strategy with

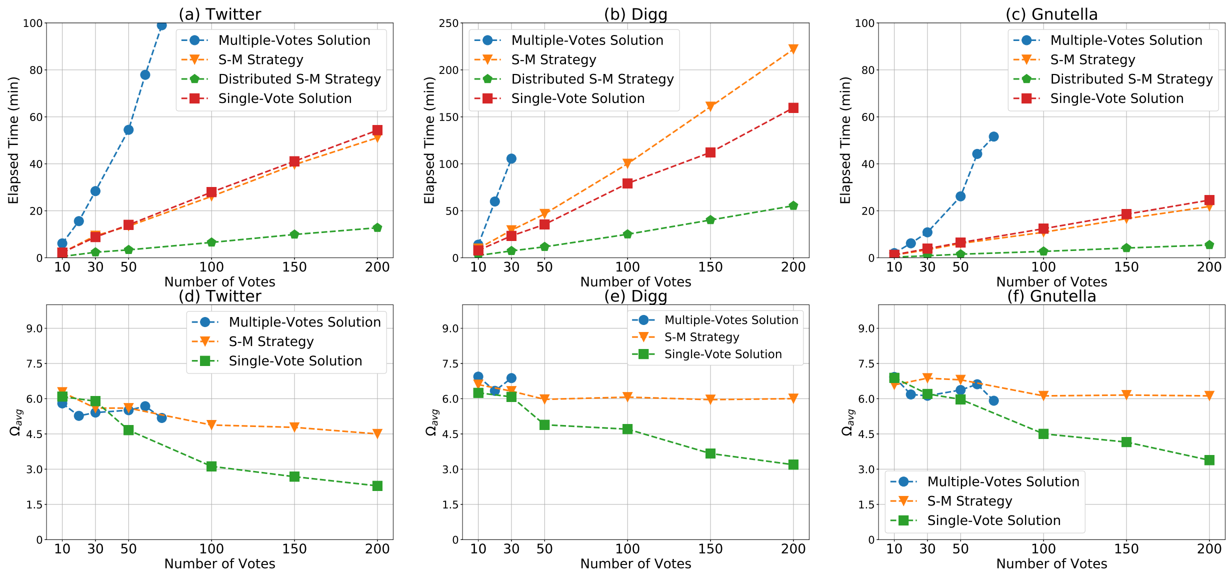
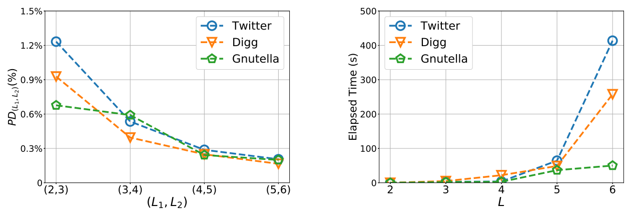
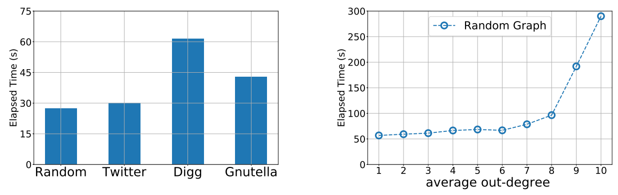


Fig. 7. The number of votes vs elapsed time and Ω_{avg} in different real-life graphs



(a) (L_1, L_2) vs $PD_{(L_1,L_2)}$ (b) L vs elapsed time

Fig. 8. Percentage difference and elapsed time for different settings of L



(a) Size of graph vs elapsed time (b) Average out-degree of graph vs elapsed time

Fig. 9. Graph characteristics vs elapsed time

five different settings: $L \in \{2, 3, 4, 5, 6\}$. To investigate the impact of L on similarity scores, we evaluate the sum of similarity scores between the query and the top-k answers, $Sum_L = \sum_{v_a \in A_k} S_L(v_q, v_a)$. The percentage-wise difference between two settings, L_1 and L_2, is defined as follows:

$$PD_{(L_1,L_2)} = \frac{Sum_{L_2} - Sum_{L_1}}{Sum_{L_1}} \tag{20}$$

The results of $PD_{(L_i,L_{i+1})}$ are shown in Fig. 8(a). We can see that the decrease of $PD_{(L_1,L_2)}$ becomes slim when L is larger than 5.

Fig. 8(b) shows the elapsed time of graph optimization for the path pruning strategy with different settings of L. The increase of L leads to an accelerated growth of elapsed time. After L becomes over 5, the computation is so costly that we cannot efficiently solve the SGP problem. Therefore, in our experiments, we evaluate the similarity between nodes by only consider the paths with a length shorter than 5 ($L = 5$).

Graph Characteristics. In the final experiment, we investigate the impact of graph characteristics on the elapsed time of graph optimization. Specifically, we study the number of nodes in the graphs and the average out-degree of the graphs. The elapsed time of graph optimization for different graphs with the same parameter settings are shown in Fig. 9(a). While the size of *Gnutella* (=62,586) is larger than the other two graphs, its corresponding elapsed time is less than *Digg*. It means the size of the graph does not impact the elapsed time of graph optimization. The reason is that the time is mainly determined by the number of the optimized edges, which is further determined by the number of the paths between the questions and the answers. In a directed graph, the number of such paths is correlated to the average out-degree of the graph, but not the number of the nodes. As such, the time cost is not necessarily increased as the graph size grows.

To gain more insights into this observation, we generate 10 random graphs with different average out-degrees to study the impact of the average out-degree. As shown in Fig. 9(b), the elapsed time of graph optimization increases significantly as the average out-degree grows to 9 or 10. This is mainly because the higher average out-degree leads to a dense graph, which increases the number of edges involved in the update and hence prolongs the optimization process.

VIII. Conclusions

In this paper, we have proposed an interactive framework to optimize the edge weights in a knowledge graph through user votes. We proposed a new notion called extended inverse P-distance to evaluate the similarity between the query node and answer nodes. This enables us to encode the user votes as constraint functions and transform the graph optimization problem into an SGP problem. Then, we developed a basic single-vote solution and a more advanced multiple-votes solution for graph optimization. Furthermore, we proposed a split-and-merge strategy to speed up the process of graph optimization for large-scale datasets. The experiment results on real-life and synthetic graphs validate the effectiveness and efficiency of our proposed framework and optimization techniques.

As for future work, we plan to extent our framework to refine the structure of a knowledge graph by adding or deleting edges. One possible solution is to determine the necessity of the existence of the edge in a knowledge graph through user votes. Moreover, we plan to leverage other implicit votes to optimize knowledge graphs, e.g., a user does not figure out the best answer but only return "the first answer is wrong". We shall incorporate such feedback commonly found in real applications into our framework.

Acknowledgement

This work was supported by National Natural Science Foundation of China (Grant No.: 61773167), NSF of Shanghai (No.: 17ZR1444900), and Research Grants Council, Hong Kong SAR, China, under projects 12200817 and 12244916. Xin Lin is the corresponding author.

References

[1] W. Cui, Y. Xiao, H. Wang, Y. Song, S.-w. Hwang, and W. Wang, "Kbqa: learning question answering over qa corpora and knowledge bases," *Proceedings of the VLDB Endowment*, vol. 10, no. 5, pp. 565–576, 2017.

[2] R. Catherine and W. Cohen, "Personalized recommendations using knowledge graphs: A probabilistic logic programming approach," in *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 2016, pp. 325–332.

[3] E. Palumbo, G. Rizzo, and R. Troncy, "entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 2017, pp. 32–36.

[4] D. Le-Phuoc, H. N. M. Quoc, H. N. Quoc, T. T. Nhat, and M. Hauswirth, "The graph of things: A step towards the live knowledge graph of connected things," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 37, pp. 25–35, 2016.

[5] M. Rotmensch, Y. Halpern, A. Tlimat, S. Horng, and D. Sontag, "Learning a health knowledge graph from electronic medical records," *Scientific Reports*, vol. 7, 2017.

[6] S. Yang, L. Zou, Z. Wang, J. Yan, and J.-R. Wen, "Efficiently answering technical questions-a knowledge graph approach." in *AAAI*, 2017, pp. 3111–3118.

[7] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 19–30, 2010.

[8] X. L. Dong and D. Srivastava, "Big data integration," *Synthesis Lectures on Data Management*, vol. 7, no. 1, pp. 1–198, 2015.

[9] D. C. Plaut *et al.*, "Experiments on learning by back propagation." 1986.

[10] H. Hu, G.-J. Ahn, and J. Jorgensen, "Multiparty access control for online social networks: model and mechanisms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 7, pp. 1614–1627, 2013.

[11] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[12] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and engineering*, vol. 8, no. 1, p. 67, 2007.

[13] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo, "A sample-and-clean framework for fast and accurate query processing on dirty data," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 469–480.

[14] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The llunatic data-cleaning framework," *Proceedings of the VLDB Endowment*, vol. 6, no. 9, pp. 625–636, 2013.

[15] M. Bergman, T. Milo, S. Novgorodov, and W.-C. Tan, "Query-oriented data cleaning with oracles," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1199–1214.

[16] A. Assadi, T. Milo, and S. Novgorodov, "Dance: Data cleaning with constraints and experts," in *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 2017, pp. 1409–1410.

[17] X. Wang, A. Meliou, and E. Wu, "Qfix: Diagnosing errors through query histories," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 1369–1384.

[18] M. Van Keulen and A. De Keijzer, "Qualitative effects of knowledge rules and user feedback in probabilistic data integration," *The VLDB Journal*, vol. 18, no. 5, p. 1191, 2009.

[19] P. Ojha and P. Talukdar, "Kgeval: Accuracy estimation of automatically constructed knowledge graphs," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1742–1751.

[20] J. Liang, Y. Zhang, Y. Xiao, H. Wang, W. Wang, and P. Zhu, "On the transitivity of hypernym-hyponym relations in data-driven lexical taxonomies." in *AAAI*, 2017, pp. 1185–1191.

[21] J. Liang, Y. Xiao, H. Wang, Y. Zhang, and W. Wang, "Probase+: Inferring missing links in conceptual taxonomies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 6, pp. 1281–1295, 2017.

[22] X. Lin, Y. Peng, B. Choi, and J. Xu, "Human-powered data cleaning for probabilistic reachability queries on uncertain graphs," *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[23] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa, "Fast and exact top-k search for random walk with restart," *Proceedings of the VLDB Endowment*, vol. 5, no. 5, pp. 442–453, 2012.

[24] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 538–543.

[25] ——, "Scaling personalized web search," in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 271–279.

[26] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An interior point algorithm for large-scale nonlinear programming," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 877–900, 1999.

[27] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[28] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Mathematical Programming*, vol. 89, no. 1, pp. 149–185, 2000.

[29] G. Xu, "Global optimization of signomial geometric programming problems," *European journal of operational research*, vol. 233, no. 3, pp. 500–510, 2014.

[30] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. IEEE, 2002, pp. 721–724.

[31] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

Appendix A
Proof of Decomposition Theorem

Decomposition Theorem. *For any* $p \in$ V,

$$\boldsymbol{r_p} = (1-c)\sum_{i=1}^{|O_{(p)}|} w_i \boldsymbol{r_{O_{i(p)}}} + c\boldsymbol{x_p}$$

Definition of Personalized PageRank:

$$\boldsymbol{v} = (1-c)\boldsymbol{A^*v} + c\boldsymbol{u} \tag{21}$$

where the $\boldsymbol{v}$ is the PPR vector, $\boldsymbol{u}$ is preference vetor, and A is an adjacency matrix.

Proof: First we rewrite Equation (21) in an equivalent form. For a given preference vector $\boldsymbol{u}$, we define the *derived matrix* $\boldsymbol{A}_u$ as

$$\boldsymbol{A}_u = (1-c)\boldsymbol{A}^* + c\boldsymbol{U} \tag{22}$$

where $\boldsymbol{U}$ is $n \times n$ matrix with $U_{ij} = u_i$ for all i, j. If we require that $|\boldsymbol{v}| = 1$, we can write Equation (21) as

$$\boldsymbol{v} = \boldsymbol{A_u v} \tag{23}$$

Without loss of generality, let the out-neighbors of p be $1, ..., k$. Let $\boldsymbol{A_p}$ be the derived matrix corresponding to $\boldsymbol{x_p}$ and let $\boldsymbol{A_1, ..., A_k}$ be the derived matrices for $\boldsymbol{u} = \boldsymbol{x_1, ..., x_k}$, respectively. Let $\boldsymbol{U_p}$ and $\boldsymbol{U_1, ..., U_k}$ be the corresponding $\boldsymbol{U}$'s in Equation (22). Let

$$\boldsymbol{v_p} = (1-c)\sum_{i=1}^{k} w_i \boldsymbol{r_i} + c\boldsymbol{x_p}$$

Clearly, $|\boldsymbol{v_p}| = 1$. We need to show that $\boldsymbol{A_p v_p} = \boldsymbol{v_p}$, in which case $\boldsymbol{v_p} = \boldsymbol{r_p}$, since PPV's are unique. First we have that:

$$\begin{aligned}\boldsymbol{A_p v_p} =& \boldsymbol{A_p}((1-c)\sum_{i=1}^{k} w_i \boldsymbol{r_i} + c\boldsymbol{x_p}) \\ &= (1-c)\sum_{i=1}^{k} w_i \boldsymbol{A_p r_i} + c\boldsymbol{A_p x_p}\end{aligned} \tag{24}$$

Using the identity

$$\boldsymbol{A_p} = \boldsymbol{A_i} - c\boldsymbol{U_i} + c\boldsymbol{U_p}$$

and

$$\sum_{i=1}^{k} w_i = 1$$

we have:

$$\begin{aligned}\boldsymbol{A_p v_p} =& (1-c)\sum_{i=1}^{k} w_i(\boldsymbol{A_i} - c\boldsymbol{U_i} + c\boldsymbol{U_p})\boldsymbol{r_i} + c\boldsymbol{A_p x_p} \\ &= (1-c)\sum_{i=1}^{k} w_i \boldsymbol{A_i r_i} - (1-c)c\sum_{i=1}^{k} w_i \boldsymbol{U_i r_i} \\ &\quad + (1-c)c\sum_{i=1}^{k} w_i \boldsymbol{U_p r_i} + c\boldsymbol{A_p x_p} \\ &= (1-c)\sum_{i=1}^{k} w_i \boldsymbol{r_i}^{*1} - (1-c)c\sum_{i=1}^{k} w_i \boldsymbol{x_i} \\ &\quad + (1-c)c\sum_{i=1}^{k} w_i \boldsymbol{x_p} + c\boldsymbol{A_p x_p} \\ &= (1-c)\sum_{i=1}^{k} w_i \boldsymbol{r_i} - (1-c)c\sum_{i=1}^{k} w_i \boldsymbol{x_i} + (1-c)c\boldsymbol{x_p} \\ &\quad + c((1-c)\boldsymbol{A}^* + c\boldsymbol{U_p})^{*2}\boldsymbol{x_p} \\ &= (1-c)\sum_{i=1}^{k} w_i \boldsymbol{r_i} - (1-c)c\sum_{i=1}^{k} w_i \boldsymbol{x_i} + (1-c)c\boldsymbol{x_p} \\ &\quad + (1-c)c\boldsymbol{A}^*\boldsymbol{x_p} + c^2\boldsymbol{x_p}^{*3} \\ &= (1-c)\sum_{i=1}^{k} w_i \boldsymbol{r_i} + (1-c)c\boldsymbol{x_p} + c^2\boldsymbol{x_p} \\ &\quad + (1-c)c(\boldsymbol{A}^*\boldsymbol{x_p} - \sum_{i=1}^{k} w_i \boldsymbol{x_i})^{*4} \\ &= (1-c)\sum_{i=1}^{k} w_i \boldsymbol{r_i} + (1-c)c\boldsymbol{x_p} + c^2\boldsymbol{x_p} \\ &= (1-c)\sum_{i=1}^{k} w_i \boldsymbol{r_i} + c\boldsymbol{x_p} \\ &= \boldsymbol{v_p}\end{aligned} \tag{25}$$

*1: according to Equation (23)

$$\boldsymbol{A_i r_i} = \boldsymbol{r_i}$$

*2: according to Equation (22)

$$\boldsymbol{A}_p = (1-c)\boldsymbol{A}^* + c\boldsymbol{U_p}$$

*3: according to the multiplication of matrices. $\boldsymbol{U_p}$ is $n \times n$ matrix with $U_{ij} = u_i = x_i$

$$\boldsymbol{U_p x_p} = \boldsymbol{x_p}$$

*4: derived Matrix $\boldsymbol{A}^*$ multiply $\boldsymbol{x_p}$ equal to neighbor nodes $\boldsymbol{x_i}$ multiply transfer probability w_i

$$\boldsymbol{A}^*\boldsymbol{x_p} = \sum_{i=1}^{k} w_i \boldsymbol{x_i}$$

Appendix B
Extentioned Inverse P-distance

Inspired by [25], we related the personalized PageRank scores to *inverse P-distance* in the weighted graph. Let $p, q \in V$, we define the *extended inverse P-distance* $\Phi(p,q)$ from p to q as

$$\Phi(p,q) = \sum_{z:p\rightsquigarrow q} P[z]c(1-c)^{l(z)} \tag{26}$$

where the summation is taken over all tours z (paths that may contain cycles) starting at p and ending at q, possibly touching p or q multiple times. For a tour $z = \langle v_q, ..., v_k\rangle$, the lengh $l(z)$ is $k-1$, the number of edges in z, The $P[z]$, which should be interpreted as"the probability of traveling z", is defined as $\prod_{i=1}^{k-1} w_i$ or 1 if $l(z)=0$. Now we prove that: $\Phi(p,q) = r_p(q)$.

Relation to Personalized PageRank

The relation between the extended inverse P-distances and personalized PageRank scores is given by the following theorem.

Theorem. *For all* $p, q \in V$,

$$r_p(q) = \Phi(p,q)$$

Proof: Writing the Decomposition Theorem in scalar form for page p, we get a set of n equations, one for each $q \in V$, of the form

$$r_p(q) = \begin{cases} (1-c)\sum_{i=1}^{|O(p)|} w_i r_{O_i(p)}(q) & \text{if } p \neq q, \\ (1-c)\sum_{i=1}^{|O(p)|} w_i r_{O_i(p)}(q) + c & \text{if } p = q \end{cases}$$

By a proof very similar to that given in [24], it can be shown these equations have a unique solution, so we need only show that $\Phi(p,q)$ satisfies these equations as well.

Clearly, if there is no path from p to q, then $r_p(q) = \Phi(p,q) = 0$, so suppose q can be reached from p. Consider the tours z starting at p and ending at q in which the first step is to the out-neighbor $O_z(p)$.

(1)If $p \neq q$, there is a one-to-one correspondence between such z and tours z' from $O_x(p)$ to q: for each z' we may derive a corresponding z by appending the edge $\langle p, O_x(p)\rangle$ at the beginning. Let Z be the bijection that takes each z' to the corresponding z. If the length of z' is l, then the length of $z = Z(z')$ is $l+1$. Moreover, the probability of traveling z is $P[z] = w_i P[z']$.

We can now split the sum in Equation (26) according to the first step of the tour z to write

$$\begin{aligned} r_p(q) &= \sum_{x=1}^{|O(p)|} \sum_{O_x(p)\rightsquigarrow q} P[T(z')]c(1-c)^{l(T(z'))} \\ &= \sum_{x=1}^{|O(p)|} \sum_{O_x(p)\rightsquigarrow q} w_i P[z']c(1-c)^{l(z')+1} \\ &= (1-c)\sum_{x=1}^{|O(p)|} w_i \sum_{O_x(p)\rightsquigarrow q} P[z']c(1-c)^{l(z')+1} \\ &= (1-c)\sum_{i=1}^{|O(p)|} w_i r_{O_i(p)}(q) \end{aligned} \tag{27}$$

(2) If $p = q$, then the same correspondence holds except that there is an extra tour z from p to $q = p$ which does not correspond to any tour z' starting from an $O_z(p)$: the zero length tour $z' = \langle p\rangle$. The length of this tour is 0, and in this case $P[z]c(1-c)^{l(z)} = c$. Thus, when $p = q$

$$r_p(q) = (1-c)\sum_{i=1}^{|O(p)|} w_i r_{O_i(p)}(q) + c$$