Optimizing Knowledge Graphs through Voting-based User Feedback

Ruida Yang [†1], Xin Lin [†2], Jianliang Xu [‡3], Yan Yang [†2], Liang He [†≀2]

[†] *School of Computer Science and Technology, East China Normal University, Shanghai, China*
[‡] *Department of Computer Science, Hong Kong Baptist University, Hong Kong, China*
[≀] *Shanghai Institute of Intelligent Science and Technology, Tongji University*
[1] rdyang@ica.stc.sh.cn, [2] {xlin, yanyang, lhe}@cs.ecnu.edu.cn, [3] xujl@comp.hkbu.edu.hk

***Abstract*—Knowledge graphs have been used in a wide range of applications to support search, recommendation, and question answering (Q&A). For example, in 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 Second, the knowledge graph may become out of date and cannot quickly adapt to new knowledge.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 multi-vote solution for graph optimization. We also propose a split-and-merge optimization strategy to scale up the multi-vote solution. Extensive experiments based on real-life and synthetic graphs demonstrate the effectiveness and efficiency of our proposed framework.**

***Index Terms*—Knowledge Graphs, Question Answering, Data Cleaning, Query Processing**

I. Introduction

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

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 inference of the relationships between two objects (e.g., the hyperlinks between web pages) [6] or the statistical information [5]. However, such methods are often vulnerable to source data errors or statistical errors [7]. Even worse, the relationships may become out of date as the knowledge evolves over time.

To address the aforementioned issues, inspired by the Human-In-The-Loop (HITL) model and Back Propagation (BP) [8], in this paper we propose an interactive framework that refines and optimizes edge weights of knowledge graphs through voting-based user feedback. As shown in Fig. 1(a), upon receiving the user's question, the Q&A system returns a ranked list of answers $A = \langle a_1, a_2, a_3 \rangle$ based on similarity evaluation via a knowledge graph. Specifically, following a variant of the PageRank algorithm [9], we perform random walks starting from the query node. The similarity between the query and an answer is the probability of the random walks reaching the answer (a formal definition will be given in Section III). After the user gets the ranked answers, he/she can check them and vote for the best answer. For example, if the user finds the second answer a_2 is most helpful to the question, he/she can vote this answer for the best one, suggesting that a_2, instead of a_1, should be ranked at the top. With such user feedback, we may adjust the edge weights (e.g., from Fig. 1(a) to Fig. 1(b)) so that a_2 will be ranked higher next time if a similar question is asked.

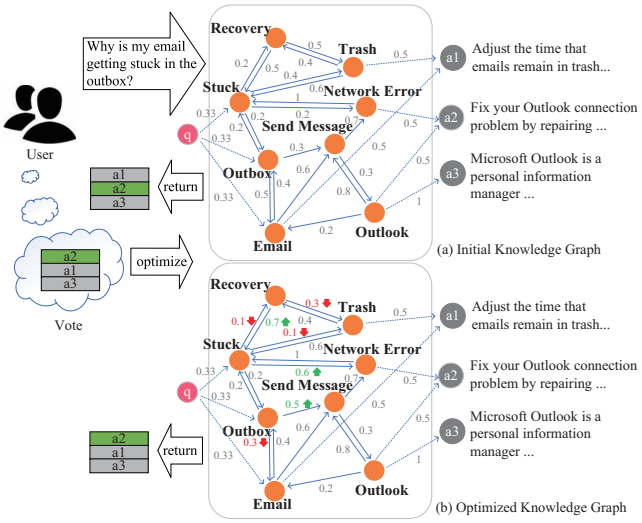


Fig. 1. An example of optimizing the knowledge graph based on user votes

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: *Consider an e-commence website. Given a query, the website recommends related products to the cus-*

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00043

tomer based on their similarities in a co-purchase knowledge graph. If it is found that customers are more likely to purchase the products that do not rank first in the recommendation list, we may want to optimize the graph with such implicit user voting information.

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

There are a number of challenges in optimizing knowledge graphs through voting-based user feedback. First, a knowledge graph is usually at large scale with complex structures, which entail update of a large number of edges in response to a user vote. Second, edge weights are continuous numerical values and hence 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 memory 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 [9], 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 [11]. To address the third 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 voting-based user 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 multi-vote solution for graph optimization. We also propose a split-and-merge strategy to optimize the multi-vote 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 multi-vote 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 multi-vote 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, graph similarity measurement, and question answering systems.

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. Bergman *et al.* [12] and Assadi *et al.* [13] employ experts with an optimized method to prune the errors in query results. Several existing works, e.g. [14], leverage human's feedback to repair the errors and hence improve the quality of data. Learn-to-rank is another technique that takes advantage of user feedback to improve the ranking accuracy. Radlinski *et al.* [15] use clickthrough data to learn ranked retrieval functions for web search results. Joachims *et al.* [16] propose an unbiased learn-to-rank algorithm with implicit user feedback. Wang *et al.* [17] transform the data cleaning problem into a mixed integer linear program (MILP) problem, which encodes user complaints about data errors. Our work is inspired by these works to integrate user feedback for data cleaning. However, the problem targeted in this paper requires the system automatically finding the edges to be optimized in a knowledge graph as well as addressing the conflicts among different user votes. The existing algorithms cannot be adapted to solving the problem proposed in this paper.

A knowledge graph is an effective way to represent the knowledge of the objective world [5]. Recently, Yang *et al.* [5] propose a novel method to answer technical questions based on a knowledge graph. Some works have been proposed to clean the knowledge graph. Liang *et al.* [18] derive new *isA* relationships according to the transitivity of the *isA* relation in the knowledge graph. They further propose to supplement the *isA* relationship by using the collaborative filtering method [19]. Lin *et al.* [20] suggest cleaning a probabilistic graph for reachability queries via crowdsourcing. Mitchell *et al.* [21] also utilize human feedback to optimize a knowledge graph. But different from our proposal, they require humans to directly evaluate the *beliefs* represented in the knowledge graph. In contrast, our framework leverages users' application-level feedback to optimize the knowledge graph.

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) [22], Personalized PageRank (PPR) [9], and SimRank (SR) [23]. 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.

Query Answering (Q&A) Systems. With the development of deep learning, several end-to-end models on Q&A have been proposed. Zhang *et al.* [24] design a novel attentive interactive neural network (AI-NN) to highlight the text segments useful to answer selection. Tan *et al.* [25] combine CNN and RNN to extract linguistic information from both structures. Zhao *et al.* [26] consider deep semantic relevance between question-answer pairs and the answers' authority. Shen *et al.* [27] match queries and answers by combining lexical and

sequential information. A knowledge base (KB) is also an important part of some Q&A systems. Hao *et al.* [28] integrate the rich KB information into the representation of the answers. Das *et al.* [29] employ memory networks to attend to the facts in the combination of text and KB. However, these end-to-end models lack interpretability, which limits their application.

III. Preliminary and Problem Definition

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

A. Preliminary

Knowledge Graph. We denote a knowledge graph as $G = (V, E, W)$, where V and E are the sets of nodes and edges, respectively. In our problem, the nodes represent the entities in the knowledge graph and the edges are their relations. For a pair of nodes v_i and v_j, $w(v_i, v_j) \in W$ is the weight of the directed edge connecting v_i to v_j. For example, as illustrated in Fig. 1, the entities in a Q&A system are technical terms or events, which can be extracted from the answer documents by using the sequential labelling method [5]; each weight $w(v_i, v_j)$ indicates the entities' semantic relevance, which can be initialized with the conditional probability of node v_j given node v_i, *i.e.*, $w(v_i, v_j) = P(v_j|v_i) = \frac{\#(v_i, v_j)}{\#(v_i)}$, where $\#(v_i)$ denotes the occurrence frequency of the entity represented by v_i and $\#(v_i, v_j)$ denotes the co-occurrence frequency of the entities of v_i and v_j in the answer documents [5].

For ease of presentation, we regard the queries and answers as query nodes and answer nodes, respectively. The set of query nodes is denoted by $Q = \{v_{q_1}, v_{q_2}, \cdots, v_{q_n}\}$ and the set of answer nodes is denoted by $A = \{v_{a_1}, v_{a_2}, \cdots, v_{a_n}\}$. Note that Q and A are linked to the knowledge graph G, but $Q \cap V = \emptyset$ and $A \cap V = \emptyset$. Formally, given a query node v_q and an entity node v_i, the weight between v_q and v_i is defined by the occurrence frequency of v_i in the query q, *i.e.*, $w(v_q, v_i) = \frac{\#(q, v_i)}{\sum_{v_j \in V} \#(q, v_j)}$, where $\#(q, v_i)$ denotes the times of the entity of v_i occurring in q. For example, in Fig. 1, the weights from the query node v_q to the nodes "Stuck", "Outlook", and "Email" are all equal to 0.33. The weights between the answer nodes and the entity nodes are derived in the same way.

In the sequel, we consider the augmented knowledge graph that combines the original graph G with Q and A for question answering.

Personalized PageRank. Personalized PageRank (PPR) [9], a variant of PageRank, is commonly used to measure the similarity between a query and the possible answers in a knowledge graph.

Consider an augmented knowledge graph $G = (V, E, W)$. Let $\boldsymbol{M}$ be the adjacency matrix of the edge weights in the graph G, where $\boldsymbol{M}_{ij} = w(v_j, v_i)$. W.l.o.g., we number the nodes in G from 1 to $|V|$, where $|V|$ is the number of nodes in the graph. According to the definition of PPR in [30], given a target query node v_q, the PPR 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 PPR vector $\boldsymbol{\pi}_{v_q}$ is of length $|V|$, where π_{v_q, v_i} denotes the i-th component of $\boldsymbol{\pi}_{v_q}$. In addition, $\boldsymbol{u}_{v_q}$ is a *preference vector*, where u_{v_q, v_i} denotes the amount of preference for node v_i, and $\sum_1^{|V|} u_{v_q, v_i} = 1$. Since we want to measure the similarity between the target query node v_q and the answer nodes, v_q is the only preference node. Therefore, we set $u_{v_q, v_i} = 1$ if $v_i = v_q$ and $u_{v_q, v_i} = 0$ if $v_i \neq v_q$.

TABLE I
Frequently Used Notations

Notation	Meanings
$S(v_q, v_a)$	The similarity between v_q and v_a
t_-; t_+	The negative vote and the positive vote
T_-; T_+	The negative vote set and the positive vote set
G^*	The optimized graph
$\Omega(G^*)$	The score of the graph G^*
$w(v_i, v_j)$	The weight of edge from node v_i to node v_j
$v_{q_{t_-}}$; $v_{q_{t_+}}$	The query nodes in t_- and t_+
$A(v_{q_{t_-}})$; $A(v_{q_{t_+}})$	The top-k answer lists of $v_{q_{t_-}}$ and $v_{q_{t_+}}$
$a^*_{t_-}$; $a^*_{t_+}$	The best answers of t_- and t_+

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

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

where typically $c \approx 0.15$, and studies have shown that small changes in c have little effect on the results [9]. A solution $\boldsymbol{\pi}_{v_q}$ to Equation (1) is a steady-state distribution of similarity measures between v_q and the 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 [11], 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

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

Here coefficients $c_{i_k} \in \boldsymbol{R}$ and $e_{i_{jk}} \in \boldsymbol{R}$, 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

This section presents the problem to be studied in this paper. For better readability, we summarize the frequently used notations in Table I.

We start by introducing how to evaluate the similarities between a query and the answers based on a knowledge graph.

Definition 1: **Query-Answer Similarity Measure.** Given an augmented knowledge graph $G = (V, E, W)$, a query node v_q, and an answer node v_a, the similarity between v_q and v_a, denoted as $S(v_q, v_a)$, is derived based on the edge weights of the graph. Following the definition of Personalized PageRank (PPR) discussed in the last section, $S(v_q, v_a)$ is defined as:

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

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

For each query q, we assume that our framework returns a ranked list of top-k answers $A_k = \langle a_1, a_2, \cdots, a_k \rangle$, which is sorted by the similarity score $S(v_q, v_a)$.

Definition 2: **Negative Vote and Positive Vote.** For each different query, the returned answer list may receive a vote from the query user. 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. We use T_+ and T_- to denote the set of positive votes and the set of negative votes, respectively.

Problem Statement. Given a set of different queries and their associated votes, the objective of our problem is to adjust the edge weights of the knowledge graph so that the best answers voted by users can be re-ranked as high as possible. Let a_t^* be the best answer voted in a vote t and $G^* = (V, E, W^*)$ denote the knowledge graph with the adjusted edge weights. Considering the potential conflicts between the votes, the objective of our problem is formalized as follows.

Definition 3: **Optimization Objective.** Given a set of negative votes T_- and a set of positive votes T_+, the score of the graph G^* is defined as:

$$\Omega(G^*) = \sum_{t \in T_- \cup T_+} (rank_t - rank'_t). \tag{5}$$

where $rank_t$ is the position of $v_{a_t^*}$ in the answer list computed with the original graph G, 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 computed with the refined G^*, e.g., if $v_{a_t^*}$ ranks first in the re-ranked list, $rank'_t = 1$. Based on this definition, the optimization objective is to find a G^* such that

$$G^* = \arg\max(\Omega(G^*)) \tag{6}$$

More intuitively, we seek the maximum increase of the ranking of the best answers for all negative votes $t_- \in T_-$ and the minimum decrease of the ranking of the best answers for all positive votes $t_+ \in T_+$. 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 [9], which would incur prohibitively high time complexity. Furthermore, the solution of Equation (1) presents the similarity scores between a target node and all other nodes in the graph, which is not necessary for our question answering system.

Inspired by [23] [30], we propose an *extended inverse P-distance*, a notion based on *inverse P-distance* that was introduced in [30]. The extended inverse P-distances are equivalent to the PPR vector scores, which can evaluate the similarity between a query node and any answer node. Formally, we define the extended inverse P-distance $\Phi(v_q, v_a)$ from v_q to v_a as:

$$\Phi(v_q, v_a) = \sum_{z: v_q \rightsquigarrow v_a} P[z] c (1-c)^{|z|} \tag{7}$$

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:

$$P[z] = w(v_q, v_1) w(v_k, v_a) \prod_{i=1}^{k-1} w(v_i, v_{i+1}), i = 1, 2, \cdots \tag{8}$$

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 PPR vector 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 PPR vector scores. Second, we prove that the extended inverse P-distance is equivalent to the PPR score derived from the decomposition theorem. The full proof is presented in [31]. □

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

$$S(v_q, v_a) = \Phi(v_q, v_a) \tag{9}$$

Example: Consider the knowledge graph shown in Fig. 1(a). In order to compute $S(v_q, v_{a_3})$, we first search all paths from v_q to v_{a_3}: $\langle v_q \rightharpoonup v_{Outbox} \rightharpoonup v_{Email} \rightharpoonup v_{Send\,Message} \rightharpoonup v_{Outlook} \rightharpoonup v_{a_3}; v_q \rightharpoonup v_{Outbox} \rightharpoonup v_{Send\,Message} \rightharpoonup v_{Outlook} \rightharpoonup v_{a_3}; v_q \rightharpoonup v_{Email} \rightharpoonup v_{Outbox} \rightharpoonup v_{Send\,Message} \rightharpoonup v_{Outlook} \rightharpoonup v_{a_3}; v_q \rightharpoonup$

$v_{Email} \rightharpoonup v_{Send\,Message} \rightharpoonup v_{Outlook} \rightharpoonup v_{a_3};\rangle$. We then use Equation (9) to compute the similarity $S(v_q, v_{a_3})$ as follows:

$$\begin{aligned} S(v_q, v_{a_3}) =& (0.33 * 0.3 * 0.6 * 0.3 * 1) * c * (1-c)^5 \\ &+ (0.33 * 0.5 * 0.3 * 1) * c * (1-c)^4 \\ &+ (0.33 * 0.4 * 0.5 * 0.3 * 1) * c * (1-c)^5 \\ &+ (0.33 * 0.6 * 0.3 * 1) * c * (1-c)^4 \\ &+ \cdots \end{aligned}$$

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 an edge weight $w(v_i, v_{i+1})$ is always 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 L will be discussed in Section VII-E.

Complexity Analysis. The computational complexity of $S(v_q, v_a)$ is bounded by the number of paths from v_q to v_a. Let d be the average degree of the nodes. This complexity can be estimated as O(d^L).

B. Encoding Negative Votes

In the scenario of solving a single vote, we do not consider positive votes, since in this case the best answer has been ranked first in the ranked list and there is no need to optimize the knowledge graph. Given a negative vote $t_- \in T_-$, 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 v_i to node v_j. The initial value of $x_{i,j}$ equals the edge weight $w(v_i, v_j)$ in the knowledge graph. Then, we analyze the user negative vote. Recall 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 any other answer v_a in the list. Thus, we define the constraint functions as follows:

$$s.t. \begin{cases} S(v_q, v_{a^*}) > S(v_q, v_{a_1}) \\ S(v_q, v_{a^*}) > S(v_q, v_{a_2}) \\ \cdots \\ S(v_q, v_{a^*}) > S(v_q, v_{a_{k-1}}) \end{cases} \tag{10}$$

More formally, we substitute Equation (9) for $S(v_q, v_a)$ and rewrite the inequalities in the standard format of SGP:

$$s.t. \begin{cases} \sum_{z: v_q \rightsquigarrow v_{a_1}} P[z] c (1-c)^{|z|} - \sum_{z: v_q \rightsquigarrow v_{a^*}} P[z] c (1-c)^{|z|} < 0 \\ \cdots \\ \sum_{z: v_q \rightsquigarrow v_{a_{k-1}}} P[z] c (1-c)^{|z|} - \sum_{z: v_q \rightsquigarrow v_{a^*}} P[z] c (1-c)^{|z|} < 0 \end{cases} \tag{11}$$

Algorithm 1 *Basic*: The Single-Vote Solution

Require: T_-, G
Ensure: G^*
1: $G^* \leftarrow G$
2: **for** each $t_- \in T_-$ **do**
3: // obtain the variable set X
4: $X \leftarrow ObtainVariableSet(t_-, G^*)$
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: $sgl\_cons \leftarrow GenerateConstraints(t_-)$
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^*

By satisfying the constraint functions, the problem maximizes the ranking increase of the best answer for the negative vote t_-, thereby maximizing the objective of our graph optimization problem stated in Definition 3. 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. The objective function of the SGP problem is set to minimize the amount 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 v_i to node v_j in X and X^*, respectively, $d(X, X^*)$ is defined as follows:

$$d(X, X^*) = \sum_{x_{i,j} \in X, x^*_{i,j} \in X^*} (x^*_{i,j} - x_{i,j})^2 \tag{12}$$

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_-, the edge weight variable set X and the initial weighted directed graph G, and outputs a new knowledge graph G^*.

The single-vote solution processes the negative votes sequentially in a greedy manner. As shown in Algorithm 1, for each $t_- \in T_-$, 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_- 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 graph (Lines 11-15). Finally, $NormalizeEdges$ normalizes the edge weights (Line 16).

After traversing each $t_- \in T_-$, the procedure returns an optimized graph G^*.

Example: Consider a set of votes: $\{t_-^1, t_+^2, t_-^3\}$, where t_-^1,t_-^3 are negative votes and t_+^2 is a positive vote. Algorithm 1 does not consider the positive vote t_+^2 and will encode t_-^1, t_-^3 into two separate SGP programs, and solve them one by one to greedily update the edge weights.

Complexity Analysis. Recall that our SGP program generates k-1 constraint functions (see Equation (10)). As each constraint function needs to evaluate the similarity between a query and an answer, which has a cost of $O(d^L)$ as analyzed in Section IV-A, the complexity of constructing an SGP program is $O(kd^L)$. Since one SGP program is constructed for each negative vote in T_-, the total construction cost is $O(|T_-|kd^L)$.

V. Multi-vote Solution

As discussed above, the single-vote solution encodes each negative vote $t_- \in T_-$ 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. Unfortunately, the single-vote solution 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 weights in the graph will be biased towards the last programming result, which may reduce 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 *multi-vote 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, 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 $t_+ \in T_+$, 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:

$$s.t. \begin{cases} \sum_{z:v_q \rightsquigarrow v_{a_2}} P[z]c(1-c)^{|z|} - \sum_{z:v_q \rightsquigarrow v_{a_1}} P[z]c(1-c)^{|z|} < 0 \\ \cdots \\ \sum_{z:v_q \rightsquigarrow v_{a_{k-1}}} P[z]c(1-c)^{|z|} - \sum_{z:v_q \rightsquigarrow v_{a_1}} P[z]c(1-c)^{|z|} < 0 \end{cases} \tag{13}$$

Note that these constraint functions are similar to the constraint functions (Equation (11)) generated for a negative vote.

Multi-Vote Solution. Similar to the single-vote solution, we formulate an SGP problem for the multi-vote 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 multi-vote SGP problem. Specifically, for each negative vote $t_- \in T_-$ and each positive vote $t_+ \in T_+$, we denote the questions in t_- and t_+ as $v_{q_{t_-}}$ and $v_{q_{t_+}}$, respectively. The comprehensive set of constraint functions are defined as follows (for clarity, we use the original form to represent the constraints):

$$\begin{aligned} &\forall t_- \in T_-, \forall v_{a_-} \in \{A(v_{q_{t_-}}) \backslash a^*_{v_{q_{t_-}}}\} \\ &\forall t_+ \in T_+, \forall v_{a_+} \in \{A(v_{q_{t_+}}) \backslash a_{1q_{t_+}}\} \\ &s.t. \begin{cases} S(v_{q_{t_-}}, a^*_{v_{q_{t_-}}}) > S(v_{q_{t_-}}, v_{a_-}) \\ S(v_{q_{t_+}}, a_{1v_{q_{t_+}}}) > S(v_{q_{t_+}}, v_{a_+}) \end{cases} \end{aligned} \tag{14}$$

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

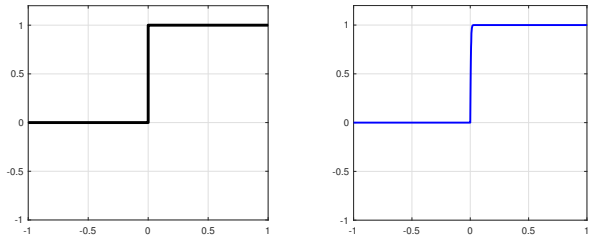
If there are no conflicts among the constraint functions of the SGP problem, it is likely that all functions in Equation (14) can be satisfied. In this case, the multi-vote problem can be easily solved by a normal SGP solver. However, there could be conflicts among the functions, due to the following reasons: (1) errors exist in some user votes so that it is unable to adjust the edge weights to satisfy the votes; (2) there could be conflicts among the user votes. As such, the functions in Equation (14) cannot be fully satisfied.

Specifically, for the first reason, the errors in a user vote refer to the user's wrong choice of the best answer. In this case, the best answer in the vote cannot gain the highest similarity to the question no matter how the weights of the graph are changed. This problem cannot be solved by optimizing the knowledge graph. Therefore, we propose an efficient judgment algorithm to filter out such user feedback that cannot be optimized because of the first reason, before we encode them into an SGP problem.

Given a question v_q, we compute its similarity with each possible answer and return a ranked list. Assuming that *rank* is the position of v_{a^*} in the ranked list, the algorithm is designed to compare whether the similarity $S(v_q, v_{a^*})$ is greater than $S(v_q, v_{a_{rank-1}})$ under an extreme condition. Specifically, we first identify all paths with a length shorter than L from node v_q to node v_{a^*} and node $v_{a_{rank-1}}$, respectively. Then, we define $Set(v_{a^*})$ and $Set(v_{a_{rank-1}})$ as the set of edges which compose the paths from node v_q to node v_{a^*} and node $v_{a_{rank-1}}$, respectively. The extreme condition is defined as follows:

- If an edge belongs to $Set(v_{a^*}) \bigcap Set(v_{a_{rank-1}})$, the weight of this edge is set to a constant greater than 0 and less than 1.
- If an edge belongs to $Set(v_{a^*}) - Set(v_{a_{rank-1}})$, the weight of this edge is set to 1.
- If an edge belongs to $Set(v_{a_{rank-1}}) - Set(v_{a^*})$, the weight of this edge is set to 0.

In other words, this condition maximizes $S(v_q, v_{a^*})$ while it minimizes $S(v_q, v_{a_{rank-1}})$. Under this condition, if $S(v_q, v_{a^*})$



(a) Step function (b) Sigmoid function

Fig. 2. The comparison of step function and sigmoid function.

can be greater than $S(v_q, v_{a_{rank-1}})$, the associated user vote will be encoded in SGP. Otherwise, the vote will be discarded.

For the second reason, conflicts among user votes may make the SGP solver unable to find any assignment of edge weights to satisfy all user votes. Since we use PPR to measure the similarity between nodes, these implicit conflicts are only reflected in the constraint functions of the SGP problem. In another words, we cannot precisely locate the causes of the conflicts to some specific edges. In fact, even locating to certain edges cannot help solve these conflicts. Hence, in this situation, we propose a method that maximizes the number of constraint functions that are satisfied, so as to minimize the influence of the conflicts.

Inspired by Goal Programming [32] and Multi-objective Optimization [33], we first introduce deviation variables into the original constraint functions of the SGP problem. The new constraint functions are as follows:

$$\begin{aligned} &\forall t_- \in T_-, \forall v_{a_-} \in \{A(v_{q_{t_-}}) \backslash a^*_{v_{q_{t_-}}}\} \\ &\forall t_+ \in T_+, \forall v_{a_+} \in \{A(v_{q_{t_+}}) \backslash a_{1 q_{t_+}}\} \\ s.t. &\begin{cases} S(v_{q_{t_-}}, v_{a_-}) - S(v_{q_{t_-}}, a^*_{v_{q_{t_-}}}) - d_{x_-} < 0 \\ S(v_{q_{t_+}}, v_{a_+}) - S(v_{q_{t_+}}, a_{1 v_{q_{t_+}}}) - d_{x_+} < 0, \end{cases} \end{aligned} \tag{15}$$

where d_{x_-} and d_{x_+} are the deviation variable for each constraint function. Let $\mathcal{D}$ denote the set of these deviation variables. For each $d_{x_i} \in \mathcal{D}$, if $d_{x_i} \leq 0$, the corresponding original constraint function, e.g., $S(v_{q_{t_-}}, v_{a_-}) - S(v_{q_{t_-}}, a^*_{v_{q_{t_-}}}) < 0$, can be satisfied. Otherwise, the function may not be satisfied.

Then, we propose a new objective function and add it to the optimization problem. The objective function should have the following two important features: (1) the function can achieve the objective of maximizing the number of constraint functions that are satisfied; (2) the function is continuous and smooth, which makes it easier for the solver to solve the optimization problem.

Intuitively, the effect of d_{x_i} being slightly larger than 0 is equivalent to that of being much larger than 0. In both cases, the corresponding constraint function may not be satisfied. Essentially, the number of unsatisfied constraint functions would be decided by the size of $|\{d_{x_i} | d_{x_i} > 0\}|$. To model this, we introduce a step function as follows:

$$\mathcal{F}(d_{x_i}) = \begin{cases} 1, d_{x_i} > 0 \\ 0, d_{x_i} \leq 0 \end{cases} \tag{16}$$

The summation of the step function $\sum_{d_{x_i} \in \mathcal{D}} \mathcal{F}(d_{x_i})$ is exactly the size of $|\{d_{x_i} | d_{x_i} > 0\}|$. However, the step function is discontinuous at $x = 0$, which would increase the difficulty of solving the programming problem. As a substitute, we choose a sigmoid function to approximate $F(d_{x_i})$. The definition of the sigmoid function is as follows:

$$\mathcal{L}(d_{x_i}) = \frac{1}{1 + e^{-w d_{x_i}}} \tag{17}$$

where w is usually set to a large integer value [34]. If d_{x_i} is larger than 0, the growth of the sigmoid function increases sharply and approaches 1 quickly. On the other hand, when d_{x_i} is less than 0, the value of the function tends to be 0. Fig. 2 shows that the sigmoid function is a close approximation of the step function when $w = 300$.

Thus, we define a new objective function as follows:

$$\sum_{d_{x_i} \in \mathcal{D}} \left(\frac{1}{1 + e^{-w d_{x_i}}}\right) \tag{18}$$

Next, we show that minimizing the above objective function is equivalent to optimizing our graph optimization problem (Definition 3). First, we prove that minimizing the size of $|\{d_{x_i} | d_{x_i} > 0\}|$ and maximizing the optimization objective in Definition 3 are equivalent. Recall from Definition 3 that we aim to enlarge the similarity between the question and the best answer. For a constraint function $S(v_q, a^*) > S(v_q, a_-) - d_{x_i}$, when $d_{x_i} \leq 0$, $S(v_q, a^*)$ is larger than $S(v_q, a_-)$. Therefore, minimizing the size of $|\{d_{x_i} | d_{x_i} > 0\}|$, which maximizes the size of $|\{d_{x_i} | d_{x_i} \leq 0\}|$, is equivalent to maximizing the optimization objective. Second, we analyze that minimizing the objective function in Equation (18) is approximately equivalent to minimizing the size of $|\{d_{x_i} | d_{x_i} > 0\}|$. As mentioned above, the sigmoid function is a close approximation of the step function. As such, the function $\sum_{d_{x_i} \in \mathcal{D}} (\frac{1}{1+e^{-w d_{x_i}}})$ closely approximates the summation of the step function, $\sum_{d_{x_i} \in \mathcal{D}} \mathcal{F}(d_{x_i})$. The latter is just the size of $|\{d_{x_i} | d_{x_i} > 0\}|$.

In the final implementation, we need to minimize both of the objective functions expressed in Equations (12) and (18). To do so, we employ a weighted summation of these two functions as the overall objective function:

$$\lambda_1 \cdot \sum_{x_{i,j} \in X, x^*_{i,j} \in X^*} (x^*_{i,j} - x_{i,j})^2 + \lambda_2 \cdot \sum_{d_{x_i} \in \mathcal{D}} \left(\frac{1}{1 + e^{-w d_{x_i}}}\right), \tag{19}$$

where λ_1 and λ_2 are the preference parameters on the graph weight changes and the degree of vote satisfaction.

Example: Revisit the example in Section IV-C, the multi-vote solution will encode all the votes t^1_-,t^2_+,t^3_- into a single SGP program, with Eq. (15) as constraint functions and Eq. (19) as objective function, and solve it to adjust the edge weights at one time.

Complexity Analysis. In contrast to the single-vote solution, the multi-vote solution employs only one SGP program to encode all the votes in T_- and T_+. Thus, it is easy to derive that the complexity of constructing the SGP program for the multi-vote solution is $O((|T_-| + |T_+|)kd^L)$, where k is the number of returned answers, d is the average degree of the nodes, and L is the pruning threshold of the path length.

VI. Optimization for Multi-Vote Solution

One issue with our multi-vote 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

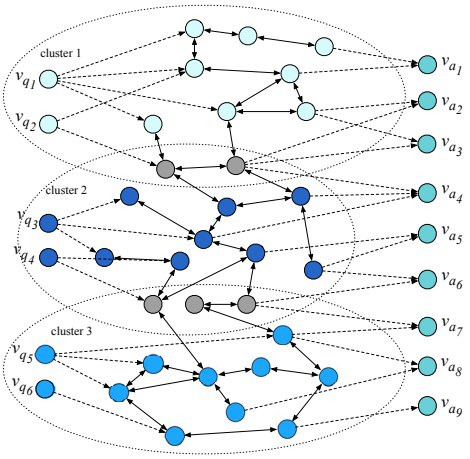


Fig. 3. An example of split strategy

constraint functions. In this section, we propose a *split-and-merge* strategy to accelerate the processing of the multi-vote solution.

A. Split and Merge

Since SGP is an NP-hard problem [35], 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. Following the approach of graph partitioning [36], 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:

$$Sim(t_i, t_j) = \frac{E(t_i) \cap E(t_j)}{E(t_i) \cup E(t_j)} \tag{20}$$

We employ an affinity propagation (AP) clustering algorithm [37] 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. 3 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 multi-vote solution to solve them separately.

We remark that the split strategy makes sense for real knowledge graphs. The nodes with high correlations centrally distributed in a sub-graph may represent a domain in a knowledge graph. For example, the entities of athletes will be distributed in the sub-graph which represents *Sports*.

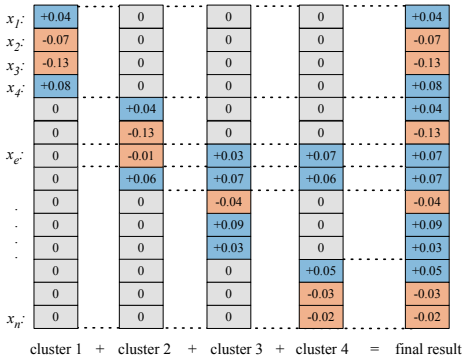


Fig. 4. An example of merge strategy

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. 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. 4 shows an example of four clusters. The changes of variable x_e are $\langle -0.01, +0.03, +0.07 \rangle$ for clusters 2, 3, and 4. 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 since $\sum_{\mathcal{C}} n_{\mathcal{C}} \Delta x^{\mathcal{C}}_{i,j} = 10 * (-0.01) + 8 * 0.03 + 9 * 0.07 \geq 0$. Therefore, we choose the maximum 0.07 of $\langle -0.01, 0.03, 0.07 \rangle$ as the final result. As can be seen, while the merge strategy we employ is simple and efficient, it tends to satisfy the results of most clusters by using the voting mechanism. The experimental results in Section VII-D confirm the effectiveness of the voting mechanism.

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 categories: 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 answers, and built a knowledge graph with 1,663 nodes and 17,591 edges.

We recruited five volunteers to conduct a user study. We first invited them to ask 100 questions. Then, we extracted the entities in each question by a Natural Language Processing tool. If these entities appear in the knowledge graph, the question will be linked to the corresponding nodes. For each question, we returned a ranked list of answers based on similarity evaluation using the knowledge graph. Then, we collected the user feedback in the form of votes, which 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: $N_Q = 100$, $N_A = 2,379$, $N_D = 1,000$, $N_{degree} = 4$, $N_{nodes} = 10,000$, $k = 20$, and $N_{aveN} = 10$.

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

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

TABLE II
STATISTICS OF GRAPH DATASETS

DataSet	$\|V\|$	$\|E\|$	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	-	-

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 II.

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 3). Ω_{avg} is defined as follows:

$$\Omega_{avg} = \frac{\sum_{t \in T_- \cup T_+} (rank_t - rank'_t)}{|T_-| + |T_+|} \tag{21}$$

  where the definitions of t, $rank_t$, and $rank'_t$ are the same as those in Definition 3. 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 multi-vote 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. For solving SGP problem in the multi-vote solution, we use *fmincon* function in MATLAB. In the paper [38], the author have proved that the complexity of *fmincon* function is O($n^{3.5}l^2$ln l ln ln$l$), where n is the number of the variables and the precision of the solution is O(l). In our setting, n equals to $|V| + |X|$, where X is the set of weights involved in the votes.

B. Effectiveness of Graph Optimization

This first set of experiments examines the effectiveness of our framework 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 merely based on the negative votes, and the multi-vote solution that is based on both the negative and positive votes. For the multi-vote solution, we set the optimization preference parameters, λ_1 and λ_2, both to 0.5. We use the test dataset to measure the effectiveness of the graph optimization.

TABLE III
SAMPLES OF OPTIMIZED EDGE WEIGHTS

Head Entity	Tail Entity	Original	Optimized	Diff
Juhuasuan	rule	0.1	0.08	-0.02
Juhuasuan	refund	0.1	0.13	0.03
cart	purchase guide	0.045	0.038	-0.007
cart	commodity	0.045	0.048	0.003

TABLE IV
RANKING OF BEST ANSWERS IN TEST DATASET

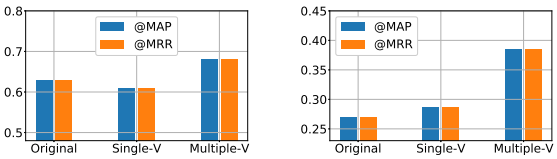
Graph	R_{avg}	Ω_{avg}	P_{avg}
Original Graph	3.56	-	-
Optimized by single-vote solution	3.59	-0.03	-0.84%
Optimized by multi-vote solution	**2.86**	**0.67**	18.82%

We list some optimized edge weights in Table III. After optimization, *cart* and *commodity* get a higher weight, and the weight between the *Juhuasuan* and *rule* becomes smaller. This is because the users rarely consulted the rules during the user study. 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 IV. The average position of the best answers in the answer list is dropped from 3.56 to 3.59 and raised to 2.87 by the basic single-vote solution and the multi-vote solution, respectively. On average, the ranking of the best answers is degraded by 0.84% and promoted by 18.82%. Clearly, the multi-vote 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 merely 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.

TABLE V
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
Q&A proposed in [5]	0.47	0.68	0.77	0.89
KG without optimization	0.49	0.69	0.79	0.90
KG optimized by single-vote solution	0.45	0.68	0.81	0.92
KG optimized by multi-vote solution	**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 without optimization) approach, which uses the extended inverse P-distance to evaluate the similarities, we include an information retrieval-based (IR) approach for comparison. The IR approach evaluates the entities in the questions and documents and returns top-k answers based on their coincidence rates. We also compare the KG-based Q&A algorithm proposed in [5], which adopts random walk to evaluate the similarity between two nodes. The results are shown in Table V. All KG approaches significantly outperform the IR approach. The performance of [5] is similar to our algorithm without any optimization, since PPR and random walk are equivalent in similarity evaluation. 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 negative user votes to adjust the edge weights. In all cases tested, the multi-vote 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. 5. MRR and MAP results of graph optimization for test dataset

Finally, Fig. 5 shows the MRR and MAP results. As shown in Fig. 5(a), the MRR and MAP degrade from 0.63 to 0.61 after optimization with the single-vote solution. In contrast, the multi-vote 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 multi-vote 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 IV and V.

TABLE VI
AVERAGE ELAPSED TIME PER QUERY

$\|A\|$	5,000	10,000	20,000	40,000
Random Walk [5]	3.0s	6.1s	13.5s	28s
Extended Inverse P-Distance	2.6s	2.8s	2.9s	3.0s

C. Efficiency of Extended Inverse P-distance

In Section IV-A, the extended inverse P-distance is proposed to efficiently evaluate the similarities between a query node and the answer nodes. We now compare its efficiency against the existing random walk method by varying the number of answers. We adopt the *linear equation group* algorithm introduced in [5] to realize the random-walk-based similarity evaluation. As shown in Table VI, the extended inverse P-distance is more efficient and scalable than the random walk algorithm. In particular, the gap increases with the size of the answer set. This is because the time cost of the random walk algorithm is linear with the number of answers, whereas the extended inverse P-distance is able to prune many lower-ranked answers by path pruning and significantly reduce the cost.

D. 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

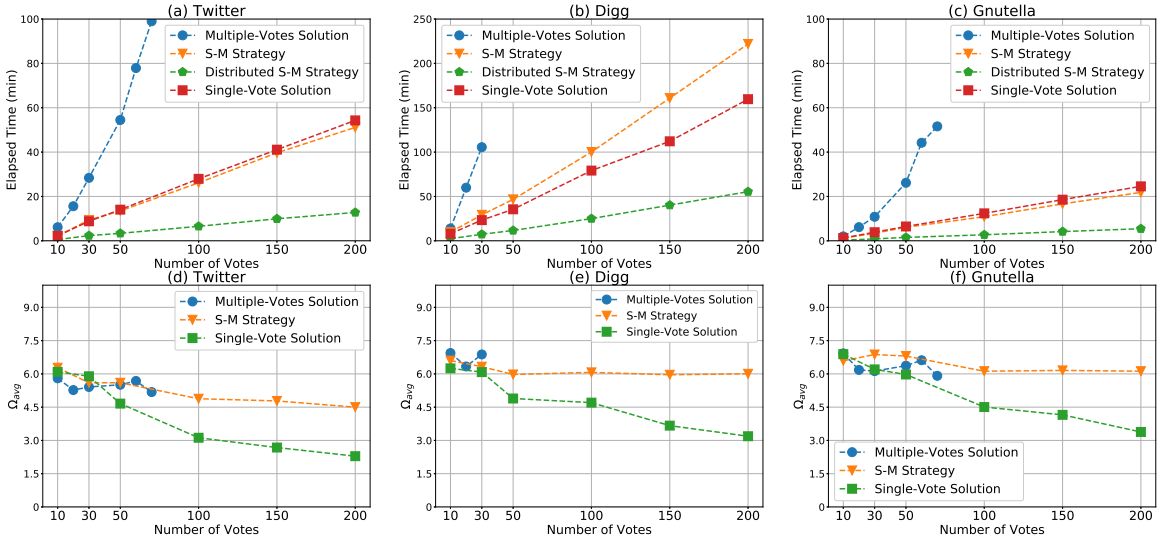


Fig. 6. The number of votes vs elapsed time and Ω_{avg} in different real-life graphs

with synthetic votes. In the experiments, we compare the elapsed time of the *basic* multi-vote solution and the multi-vote 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. 6(a)-(c), the elapsed time of the basic multi-vote 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 causes high memory consumption. 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. 6(a)).

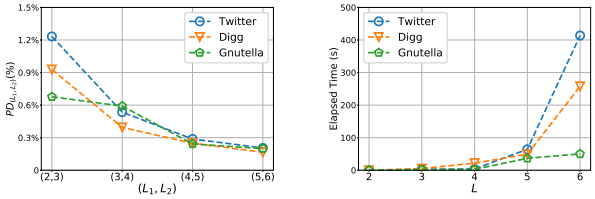
In the split-and-merge (S-M) strategy, we select the median of the similarities between votes as the classification criterion, based on which the AP algorithm automatically classifies the votes into clusters. The average size of the classified clusters is 5 votes. Compared to the basic solution, the 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. 6(d)-(f), we show the Ω_{avg} scores of different solutions. It is interesting to observe that the optimized multi-vote 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 effectiveness of graph optimization.

Regarding the single-vote solution, as shown in Figs. 6(a)-(c), it is faster than the multi-vote solution with the split-and-merge strategy. Nevertheless, the effectiveness of the multi-vote solution in graph optimization significantly outperforms the single-vote solution (Figs. 6(d)-(f)).

E. Impact of Parameter Settings

The following set of experiments is designed to justify the parameter choices. Specifically, we study the impact of the path length on the similarity scores and the execution time.



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

Fig. 7. Percentage difference and elapsed time for different settings of L

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 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_i and L_j, is defined as follows:

$$PD_{(L_i,L_j)} = \frac{Sum_{L_j} - Sum_{L_i}}{Sum_{L_i}} \tag{22}$$

The results of $PD_{(L_i,L_{i+1})}$ are shown in Fig. 7(a). We can see that $PD_{(L_i,L_{i+1})}$ becomes slim when L_i is 5.

Fig. 7(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 no longer than 5 ($L = 5$).

VIII. Conclusions

In this paper, we have proposed an interactive framework to optimize the edge weights in a knowledge graph through voting-based user feedback. 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 multi-vote solution for graph optimization. Furthermore, we proposed a split-and-merge strategy to speed up the process of graph optimization for large datasets. The experiment results on real-life and synthetic graphs validate the effectiveness and efficiency of our proposed framework and optimization techniques.

Acknowledgement

This work was supported by National Key R&D Program of China (No. 2018AAA0100503&2018AAA0100500), National Natural Science Foundation of China (No. 61773167), and Science and Technology Commission of Shanghai Municipality (No. 19511120200). Jianliang Xu's work is supported by the Research Grants Council of Hong Kong under Project Nos. C6030-18GF and 12201018. 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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

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

[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] 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.

[12] 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.

[13] 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.

[14] 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.

[15] T. Radlinski, F.and Joachimes, "Query chains: learning to rank from implicit feedback," in *KDD*, 2005, pp. 239–248.

[16] T. Joachimes, A. Swaminathan, and T. Schnabel, "Unbiased learning-to-rank with biased feedback," in *KDD*, 2017, pp. 781–789.

[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] 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.

[19] 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.

[20] 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.

[21] T. Mitchell, E. Hruschka, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, and J. Krishnamurthy, "Never-ending learning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015, pp. 2302–2310.

[22] 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.

[23] 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.

[24] X. Zhang, S. Li, and H. Sha, L.and Wang, "Attentive interactive neural networks for answer selection in community question answering," in *AAAI, San Francisco, California, USA.*, 2017, pp. 3525–3531.

[25] M. Tan, C. Santos, B. Xiang, and B. Zhou, "Improved representation learning for question answer matching," in *ACL, Berlin, Germany, Volume 1*, 2016.

[26] Z. Zhao, H. Lu, V. Zheng, D. Cai, X. He, and Y. Zhuang, "Community-based question answering via asymmetric multi-faceted ranking network learning," in *AAAI, San Francisco, California, USA.*, 2017, pp. 3532–3539.

[27] Y. Shen, W. Rong, Z. Sun, Y. Ouyang, and Z. Xiong, "Question/answer matching for CQA system via combining lexical and sequential information," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 2015, pp. 275–281.

[28] Y. Hao, Y. Zhang, L. K., S. He, Z. Liu, H. Wu, and J. Zhao, "An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1*, 2017, pp. 221–231.

[29] R. Das, M. Zaheer, S. Reddy, and A. McCallum, "Question answering on knowledge bases and text using universal schema and memory networks," in *ACL Vancouver, Canada, July 30 - August 4, Volume 2*, 2017, pp. 358–365.

[30] G. Jeh and J. Widom, "Scaling personalized web search," in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 271–279.

[31] R. Yang, 2019, https://github.com/Dontkno/OptKG.

[32] R. K. Pati, P. Vrat, and P. Kumar, "A goal programming model for paper recycling system," *Omega*, vol. 36, no. 3, pp. 405–417, 2008.

[33] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural & Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[34] A. Iliev, N. Kyurkchiev, and S. Markov, "On the approximation of the step function by some sigmoid functions," *Mathematics & Computers in Simulation*, vol. 133, pp. 223–234, 2017.

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

[36] 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.

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

[38] N. Karmarkar, "A new polynomial-time algorithm for linear programming." *Combinatorica*, vol. 4, pp. 373–395, 1984.