Hindawi
Complexity
Volume 2020, Article ID 7909826, 13 pages
https://doi.org/10.1155/2020/7909826

Research Article

Graph-Based Node Finding in Big Complex Contextual Social Graphs

Keshou Wu,[1] Guanfeng Liu ,[2] and Junwen Lu [1]

[1]*Engineering Research Center for Software Testing and Evaluation of Fujian Province, Xiamen University of Technology, Xiamen, China*
[2]*Department of Computing, Macquarie University, NSW 2109, Australia*

Correspondence should be addressed to Guanfeng Liu; guanfeng.liu@mq.edu.au

Received 26 October 2019; Accepted 18 January 2020; Published 26 February 2020

Guest Editor: Yuan Yuan

Graph pattern matching is to find the subgraphs matching the given pattern graphs. In complex contextual social networks, considering the constraints of social contexts like the social relationships, the social trust, and the social positions, users are interested in the top-K matches of a specific node (denoted as the designated node) based on a pattern graph, rather than the entire set of graph matching. This inspires the conText-Aware Graph pattern-based top-K designated node matching (TAG-K) problem, which is NP-complete. Targeting this challenging problem, we propose a recurrent neural network- (RNN-) based Monte Carlo Tree Search algorithm (RN-MCTS), which automatically balances exploring new possible matches and extending existing matches. The RNN encodes the subgraph and maps it to a policy which is used to guide the MCTS. The experimental results demonstrate that our proposed algorithm outperforms the state-of-the-art methods in terms of both efficiency and effectiveness.

1. Introduction

Graph pattern matching (GPM) is widely used in many applications, like computer vision [1], chemical structure [2], and social networks [3–6]. Formally, given a pattern graph Q and a data graph G, GPM is to compute the set $M(Q, G)$ of matches Q in G. Based on GPM, many applications focus on finding matches of a specific pattern (query) node, rather than the entire set of graph matching, for example, expert recommendation [7, 8] and egocentric search [9]. This leads to the top-K designated node matching (topKP) [10] problem, where given a designated node u_0 in Q, it is to find the top-K matching nodes of u_0 ranked by a quality function in G.

In a social network, like crowd-sourcing travel [11] and social network based e-commerce [12], social queries often need to find matches of the topKP under requirements of social contexts, i.e., the social positions, the social trust, and the social relationships that have significant influence on collaborations and decision-making [13, 14]. For example, in the expert recommendation, people are more willing to find experts who can build the intimate relationships with team members. Example 1 discusses this scenario with a social network.

Example 1. A fraction of a collaboration network is given as graph G in Figure 1. Each node in G denotes a person, with attributes such as *job title*, e.g., project manager (PM), database developer (DB), programmer (PRG), and software tester (ST). Each edge indicates a supervision relationship; e.g., edge (A_1, D_1) indicates that A_1 supervised D_1. The number added on the edge indicates the intimacy between the two persons. In order to simplify, in graph G, only the intimacy of corresponding edges of Q is given.

A company issues a graph search query to find the best PM in Q who supervises both PRG and DB, and moreover, (1) the DB worked under the PRG, and (2) the intimacy between the three persons must be larger than specific values in Q. It can be seen that, for the three PMs in G, i.e., A_1, A_2, and A_3, only A_2 does not meet the requirements. In addition, the intimacy among $\{A_3, B_2, C_2\}$ is higher than $\{A_1, B_1, C_1\}$; hence, A_3 is the PM we would like to find.

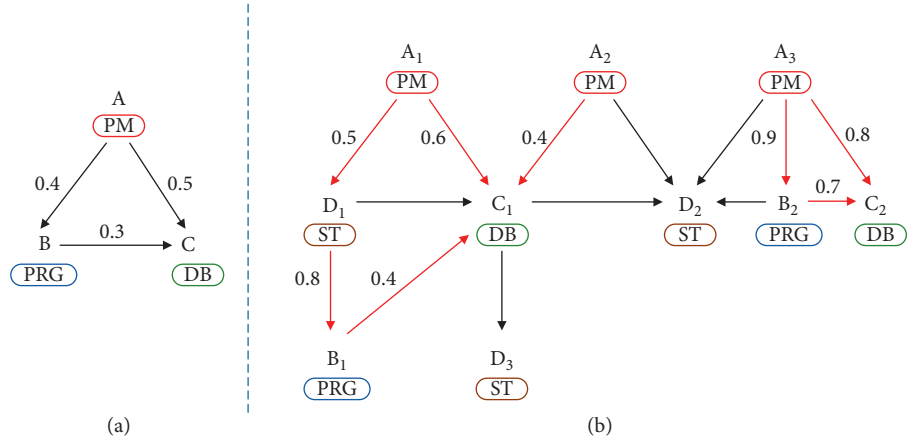

Figure 1: Querying collaboration network: (a) pattern graph Q; (b) data graph G.

This discussion above leads to the conText-Aware Graph pattern-based top-K designated node matching (TAG-K), which aims to find the topKP results under Multiconstrained Simulation (MCS) [12, 15]. There are multiple end-to-end constraints for the attributes. So the MCS becomes NP-complete, as it is exponential of the time complexity to find the best solution to investigate each combination of the different attributes. Therefore, TAG-K is NP-complete, as it subsumes the classical NP-complete multiconstrained path selection problem [12]. The existing topKP methods for the MCS problem [10, 16–19] need to enumerate all possible subgraphs that satisfy the constraints of social contexts, leading to the expensive time cost. In addition, even though these methods can find TAG-K results, once the problem statements change slightly (i.e., the value of social contexts), they need to be revised. Therefore, the existing methods are not applicable when dealing with the large-scale data graphs. In contrast, machine learning methods have the potential to be applicable across many optimization tasks by automatically discovering their own heuristics based on the training data, and they can get stable performance without adjusting the model in different social contexts.

While supervised learning is used widely in most successful machine learning methods, it is not applicable to the TAG-K problem because one does not have access to the optimal labels. However, we can compare the quality of subgraphs by using a verifier and provide some reward feedbacks to a learning algorithm. Hence, we follow the idea of Monte Carlo Tree Search (MCTS), which is an effective way of solving NP-complete problems [12, 20]. The key idea of MCTS is to construct a search tree of states (i.e., possible subgraphs) evaluated by fast Monte Carlo simulations [21]. Then, the state value is estimated as the mean outcome of the simulations. Meanwhile, a search tree is maintained to guide the direction of simulation, for which bandit algorithms can be employed to balance exploration and exploitation [22]. Then, after certain times of simulations, we can get TAG-K results based on the state values.

The problem poses a major challenge; since the state space of TAG-K is large, MCTS needs to traverse most subgraphs to get the accurate state values, which takes a lot of time. Inspired by the graph embedding technique, Hamilton and Ying [23] use neural networks to extract graph structural information and graph properties, which is an effective and efficient way to solve the graph analytic problem. In this paper, we utilize the characteristics of experience (i.e., vector representations of found matches) to train an RNN structure, which can evaluate the "potential" of the node to be the TAG-K result without traversing it. Our contributions are summarized as follows:

(i) We formulate TAG-K as the MCTS problem and design the neural network, which is trained by the subgraphs generated in the searching process to evaluate nodes in G

(ii) We propose the upper tree and the lower tree with optimal strategies to return TAG-K results without accessing all nodes in G

(iii) Using real social network datasets, we experimentally verify that our algorithm outperforms the existing methods in both efficiency and effectiveness

2. Related Work

2.1. Isomorphism-Based topKP. This type of the topKP is based on the subgraph isomorphism [24]. Tian and Patel [18] propose the concept of approximate subgraph matching, which allows node mismatches and node/edge insertions and deletions. For coping with approximate subgraph matching problem, an index-based method is presented in [18], called TALE. In addition, Ding et al. [16] define the matching similarity between a data graph and a query graph to order results. Built on NH-index in [18], Ding et al. [16] employ the index to prune unpromising candidate nodes for each query node. Furthermore, Zhu et al. [25] consider the entire structure matching rather than substructure matching and propose an algorithm to respond to the topKP similarity query using two distance lower bounds with different computational costs.

2.2. Simulation-Based topKP. Existing isomorphism-based topKP methods are still too strict to be used in some applications, e.g., finding social experts [26] and project

organization [10]. Based on graph simulation [27], Fan et al. [10] propose a novel topKP method supporting a designated pattern node u_0, which can find the topKP without computing the entire graph matching results. In addition, Chang et al. [28] study the problem of top-K tree pattern matching, where the edges in the tree are mapped to the shortest paths in G connecting the corresponding nodes and then proposed an optimal enumeration paradigm [28].

However, the existing topKP methods do not consider the social contexts which are common in social network-based applications, like crowd-sourcing travel [11, 29, 30] and social network-based e-commerce [12]. Therefore, these methods cannot support the NP-complete TAG-K that exists in many real applications.

Recently, based on [10], Lie et al. [31] propose an Monte Carlo-based approximation algorithm (MC-TAG-K) for TAG-K, which uses a random sampling method that cannot guarantee the algorithm performance, and MC-TAG-K has to traverse the data graph more than once, which costs a lot of time when the data graph is large; therefore, it is not a good solution to the TAG-K problem.

3. Preliminaries

In this section, we first introduce the data graph, pattern graph, and multiconstrained simulation (MCS) [12] and then propose the ranking function and TAG-K problem.

3.1. Data Graph. A data graph is a contextual social graph (CSG). $G = (V, E, L_G, \rho)$.

(i) V is the set of nodes of G, and each node represents the participant of a CSG.

(ii) E is the set of edges of G, and a direct edge $e \in E$ from node v to node v' is a quad (v, v', t, r), where $v, v' \in V$, $t \in [0, 1]$, denotes the *social trust* between v and v', $r \in [0, 1]$ denotes the *social intimacy degree* between v and v', the higher values of t and r, and the closer trust and social intimacy between the two participants.

(iii) L_G is a function that assigns every node in V with the social role of a specific domain.

(iv) ρ is a function that assigns every node in V with a *role impact factor*, denoted as $\rho(v) \in [0, 1]$, which illustrates the impact of the participant in a specific domain. The greater the value of $\rho(v)$, the more professional knowledge the participant v has t, r, and ρ are called *social impact factors*. For a path $P(v_1, v_n) = \langle e_1, e_2, \ldots, e_{n-1} \rangle$ in G, where $e_i = (v_i, v_{i+1}, t_i, r_i)$ is the i-th edge in $P(v_1, v_n)$. Based on the theories in *Social Psychology* [13], the aggregated social contexts of a path $P(v, v')$ are denoted as $\mathcal{A}(P(v_1, v_n)) = \{At(P(v_1, v_n)), Ar(P(v_1, v_n)), A\rho(P(v_1, v_n))\}$, where

(i) $At(P(v_1, v_n)) = \prod_1^{n-1} t_i$ is the aggregated social trust of $P(v_1, v_n)$

(ii) $Ar(P(v_1, v_n)) = \prod_1^{n-1} r_i$ is the aggregated social intimacy of $P(v_1, v_n)$

(iii) $A\rho(P(v_1, v_n)) = \sum_1^n \rho(v_i)/n$ is the aggregated role impact of $P(v_1, v_n)$

Example 2. Figure 1(b) shows the structure of a data graph, where we can find the labels and the intimacy relationships between two users.

3.2. Pattern Graph. A pattern graph is a directed graph. $Q = (V_Q, E_Q, L_Q, \lambda_Q)$, where

(i) V_Q is the set of nodes of Q

(ii) E_Q is the set of edges of Q; $(u, u') \in E_Q$ denotes the directed edge from u to u', where $u, u' \in V_Q$

(iii) L_Q is a function that assigns every node in Q with a social role

(iv) λ_Q is a function that assigns every edge (u, u') with a quad $(l, \lambda_t, \lambda_r, \lambda_\rho)$, where l is the bounded length of (u, u'), which is a positive number, and λ_t, λ_r, and λ_ρ in the range of $[0, 1]$ denote the multiple constraints of the aggregated social contexts

Example 3. Figure 1(a) shows a pattern graph that contains three nodes and the corresponding edges. In each edge, we can find one of the constraints for social intimacy between nodes.

3.3. GPM-Based Multiconstrained Simulation (MCS). A graph $G = (V, E, L_G, \rho)$ matches a pattern graph $Q = (V_Q, E_Q, L_Q, \lambda_Q)$ via MCS means there exists a binary relation $S \subseteq V_Q \times V$, where the following conditions apply:

(i) For each node $u \in V_Q$, there exists a node $v \in V$ such that $(u, v) \in S$

(ii) for each pair, $(u, v) \in S$

(a) $L_Q(u) = L_G(v)$

(b) for each edge $(u, u') \in E_Q$, there exists at least one path $P(v, v')$ (denoted as *relevant path*) v to v' in G such that $(u', v') \in S$, $|P(v, v')| \leq l(u, u')$, $|P(v, v')|$ is the number of edges in $P(v, v')$, and the aggregated social contexts meet $At(P(v, v')) \geq \lambda_t(u, u')$, $Ar(P(v, v')) \geq \lambda_r(u, u')$, and $A\rho(P(v, v')) \geq \lambda_\rho(u, u')$

It is known that if G matches Q, then there exists a unique maximum relation set $M(Q, G) = \{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ including all pairs in S.

3.4. Matches of Designated Node. Now we extend the pattern graph to be $Q = (V_Q, E_Q, L_Q, \lambda_Q, u_0)$, where u_0 is the designated node in V_Q. Then, the matches of u_0 in G is defined as $M_U(Q, G, u_0) = \{v \mid (u_0, v) \in M(Q, G)\}$.

3.5. Ranking Function

3.5.1. Relevant Set. Given a match v of a designated node u_0 in Q, the *relevant node set* of v with respect to u_0 is denoted as $R(u_0, v)$, which includes all matches v' of u' for each descendant u' of u_0 in Q; i.e., if an edge $(u_0, u') \in E_Q$, then v reaches v' via a relevant path $P(v, v')$, where $(u', v') \in M(Q, G)$.

That is, $R(u_0, v)$ includes all matches v' to which v can reach via a relevant path $P(v, v')$. Accordingly, we define *relevant edge set* $R_E(u_0, v)$ to save all relevant paths from v to all matches v', and then for any match v of a query node u_0, there exists unique, maximum relevant node set and relevant edge set.

In the node matching, we need to consider the matching of nodes and the corresponding pattern. Based on [31], the ranking function to rank matches of the designated node u_0 is defined as follows:

$$F(u_0, v) = \frac{2\alpha}{\pi}\arctan\left(\left|R(u_0, v)\right|\right) + (1-\alpha)\frac{\sum_{P_i \in R_E(u_0, v)} U(P_i)}{\left|R_E(u_0, v)\right|}, \tag{1}$$

where α is used to balance the two functions; $|R(u_0, v)|$ and $|R_E(u_0, v)|$ are the number of nodes and edges in $R(u_0, v)$ and $R_E(u_0, v)$, respectively; and $U(P_i)$ denotes the average social impact of P_i. In the complex social network, we have three social impact factors, i.e., trust, social intimacy, and social role impact,and they should be considered in the modelling of node matching. Therefore, we propose the following equation:

$$U(P) = w_t \times At(P) + w_r \times Ar(P) + w_\rho \times A\rho(P), \tag{2}$$

where w_t, w_r, and w_ρ are the weights of social impact factors t, r, and ρ, respectively; w_t, w_r, and w_ρ are in the scope of $[0, 1]$ and $w_t + w_r + w_\rho = 1$.

3.6. TAG-K Problem

Definition 1. Context-Aware Graph pattern based top-K designated node finding problem (TAG-K). Given a CSG G, a pattern graph Q with a designated node u_0, a positive integer K, TAG-K is to find a subset D, where

$$D = \underset{D'}{\operatorname{argmax}} \sum_{v_i \in D'} F(u_0, v_i), \tag{3}$$

where D' is a subset of $M_U(Q, G, u_0)$ and $|D'| = K$. That is, TAG-K is to identify a set of K matches of u_0, which maximizes the summation of ranking function $F(\cdot)$, namely, $\forall D' \subseteq M_U(Q, G, u_0)$; if $|D'| = K$, then $F(D) \geqslant F(D')$.

We denote $C(u_0)$ as the set of all the *candidates* v in G of u_0 (i.e., v has the same label as u_0) and use $F_L(u_0, v)$ and $F_U(u_0, v)$ to denote the *lower bound* and *upper bound* of $F(u_0, v)$, respectively, i.e., $F_L(u_0, v) \leq F(u_0, v) \leq F_U(u_0, v)$.

Lemma 1. *For a K-element set $D \subseteq C(u_0)$, if (1) each $v \in D$ is a match of u_0 and (2) $\min_{v \in D} F_L(u_0, v) \geq \max_{v' \in C(u_0) \setminus D} F_U(u_0, v')$, then D is the TAG-K result.*

Based on Lemma 1, we can get TAG-K results without computing the entire $M(Q, G)$.

4. Recurrent Neural Network-Based Monte Carlo Tree Search (RN-MCTS)

In our algorithm, we build the upper tree and the lower tree of each candidate node that are used to evaluate the upper bound $F_U(\cdot)$ and lower bound $F_L(\cdot)$, respectively; in the searching process, $F_U(\cdot)$ and $F_L(\cdot)$ will be updated, and $F_L(\cdot)$ will gradually close to $F(\cdot)$; hence, in the worst case, RN-MCTS can still find TAG-K results. In addition, we use the RNN structure that trained by searching results to guide MCTS and propose some optimization strategies to speed up the searching process. In this section, we first introduce the tree structure of RN-MCTS, then introduce the RNN structure, and discuss the details of RN-MCTS combined with RNN and optimization strategies.

4.1. Objective Function. Given a CSG G, a pattern graph Q, and a path $P(v, v')$ in G, we first propose the *objective function* to investigate if $P(v, v')$ meets the constraints specified for the edge in Q. We use the objective function to combine the aggregated multiple attribute values and the corresponding constrains. Then, we could investigate the value of the objective function to check the feasibility of the graph search:

$$\delta(P(v, v')) = \max\left\{\frac{\lambda_t(u_0, u')}{At(P(v, v'))}, \frac{\lambda_r(u_0, u')}{Ar(P(v, v'))}, \frac{\lambda_\rho(u_0, u')}{A\rho(P(v, v'))}, \frac{\left|P(v, v')\right|}{l(u_0, u')}\right\}, \tag{4}$$

where $u' \in Q$ and $v' \in C(u')$. From the objective function, we can see that if an edge $(u_0, u') \in Q$ can be mapped into a path $P(v, v')$ in G, $\delta(P(v, v')) \leq 1$; otherwise, $\delta(P(v, v')) > 1$.

4.2. Monte Carlo Tree Structure (MCT). Now we introduce the Monte Carlo Tree structure (MCT) of RN-MCTS, and each node in the tree represents a node in G, while the tree's edges correspond to the edges of G. MCTS grows the tree structure iteratively; with each iteration, the MCT is traversed and expanded. In an MCT, a root node represents a candidate in $C(u_0)$, and a child node represents the node connected with its ancestor by an edge in G; a leaf node represents (1) the paths through which it exceeds constraints

of social contexts or cannot be relevant paths or (2) a new expanded node that has not been visited. Each MCT can be seen as the subgraph connected to a candidate node. Each node v' in the MCT stores the set of statistics:

$$\{P(v,v'), R_{ES}(v,v'), R_S(v,v'), F_S(v,v'), N(v,v')\}. \quad (5)$$

Here, the root node v of the MCT is a candidate, $P(v,v') = \langle v,\ldots,v'\rangle$ is the path from root node v to v', $N(v,v')$ is the number of times that v' has been visited in the MCT, $R_{\mathrm{ES}}(v,v')$ is a set that saves all relevant paths that start from $P(v,v')$ in the MCT, and $R_S(v,v')$ is a set that saves the corresponding relevant nodes; i.e., for a visited relevant path $P_i = \langle v,\ldots,v',\ldots,v_i\rangle$, $P_i \in R_{\mathrm{ES}}(v,v')$ and $v_i \in R_S(v,v')$. Then, it is known $R_S(v,v')$ is a subset of $R(u_0,v)$. $F_S(v,v')$ is defined as follows:

$$F_S(v,v') = \frac{2\alpha}{\pi}\arctan\left|R_S(v,v')\right| + (1-\alpha)\frac{\sum_{P_i\in R_{\mathrm{ES}}(v,v')}U(P_i)}{\left|R_{\mathrm{ES}}(v,v')\right|}, \quad (6)$$

where α is the same factor of equation (1). In the searching process, if RN-MCTS finds a new relevant path $P(v,v_j)$ passing through $P(v,v')$, $P(v,v_j)$ and v_j will be added in $R_{\mathrm{ES}}(v,v')$ and $R_S(v,v')$, respectively, then updates $F_S(v,v')$ by (6).

Given a candidate v of a designated node u_0 in Q, if all the relevant paths of $R_E(u_0,v)$ has been visited in the MCT of v. Intuitively, $F(u_0,v)$ can be calculated as follows:

$$F(u_0,v) = \frac{2\alpha}{\pi}\arctan\left|\sum_{v_i}R_S(v,v_i)\right| + (1-\alpha)\frac{\sum_{v_i}\sum_{P_j\in R_{\mathrm{ES}}(v,v_i)}U(P_j)}{\sum_{v_i}\left|R_{\mathrm{ES}}(v,v_i)\right|}. \quad (7)$$

Here, v_i is the child node of v in the MCT. That is, after certain times of iterations, $F(\cdot)$ of a candidate can be approximated by statistics of its child nodes. Then, we formulate the TAG-K problem as the Monte Carlo Tree Search (MCTS) problem. Since $F(\cdot)$ of the root node is proportional to $F_S(\cdot)$ of child nodes, the aim of MCTS is to find high values of $F_S(\cdot)$ in the MCTs of candidates under limited searching time.

4.3. Recurrent Neural Network Policy. Now, we introduce the RNN policy. In the searching process, RNN is used to help RN-MCTS to select child nodes. Our RNN structure is simple and consists of two layers: an RNN layer and a fully connected layer. In the MCT of v_1, for the visited path $P(v_1,v_n) = \langle v_1, v_2, \ldots, v_n\rangle$, we can get the sequence: $\mathcal{T}(P(v_1,v_n)) = \langle \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{n-1}\rangle$, where $\mathcal{T}_i = \{\mathcal{P}(v_1, v_{i+1}), R_S(v,v_{i+1}), F_S(v,v_{i+1})\}$, $i \in [1, n-1]$. Here, each node v in $\mathcal{P}(v_1,v_i)$ and $R_S(v,v_i)$ is represented by a vector representation, i.e., $\mathcal{P}(v_1,v_i) = \langle \mathscr{E}(v_1), \mathscr{E}(v_2), \ldots, \mathscr{E}(v_i)\rangle$. $\mathscr{E}(v)$ is the vector representation of v that is generated by DeepWalk [32]. For the path $P(v_1,v_n)$, we set $\mathcal{T}(P(v_1,v_{n-1}))$ as the input and the node selection $\mathscr{E}(v_n)$ and $F_S(v,v_n)$ as the output to train the RNN structure, and the lost function of a time step is defined as follows:

$$\mathrm{lost}(\mathcal{T}_i,\mathcal{T}_i') = \mu\mathrm{RMSE}(\mathcal{P},\mathcal{P}') + (1-\mu)\mathrm{RMSE}(R_S,R_S') + (F_S - F_S')^2. \quad (8)$$

Here, $\mathcal{T}_i' = \{\mathcal{P}', R_S', F_S'\}$ is the prediction of the last time step $\mathcal{T}_{i-1}$. RMSE is the function of root mean squared error:

$$\mathrm{RMSE}(y,y') = \frac{\sum_{i=1}^{n}(y_i - y_S')^2}{n}. \quad (9)$$

Then, for the current searching path P, we define the output of the RNN structure (i.e., $\mathscr{E}(v_n)$ and $F_S(v,v_n)$) as $\mathscr{E}_{\mathrm{net}}(P)$ and $F_{\mathrm{net}}(P)$, respectively.

4.4. UCT Function. An important problem of MCTS is to balance the exploration versus exploitation. The exploration approach promotes the exploration of unvisited nodes in the MCT, and this means that the exploration will expand the tree's breath more than its depth. Exploitation tends to stick to one path that has the greatest estimated value (i.e., the maximal value of $F_S(\cdot)$) to find more relevant nodes. The balance of exploration and exploitation can ensure our algorithm is not overlooking any potential relevant paths, avoiding the inefficiency in the search with a large number of candidates.

Specifically, on each RN-MCTS iteration, the search process is rolled out by selecting child nodes according to the proposed variant of UCT [33] from the root node. During the search of a Monte Carlo Tree, we need to consider both the width and the depth of the tree search. Therefore, we propose equations (10) and (11):

$$v_d = \arg\max_{v_s}\left\{F_S(v,v_s) + \beta\times d\left(P\left(v,v_f\right),v_s\right)F_{\mathrm{net}}\left(P\left(v,v_f\right)\right) + \gamma\sqrt{\frac{\ln\left(N\left(v,v_f\right)\right)}{N(v,v_s)+1}}\right\}, \quad (10)$$

where γ and β are the constants that control the level of exploration, $P(v,v_f)$ is the current search path, and v_s is the child node of v_f. $d(P(v,v_f),v_s)$ is the similarity between v_s and the predict of RNN, which is defined as follows:

$$d\left(P\left(v,v_f\right),v_s\right) = \cos\left(\mathscr{E}_{\mathrm{net}}\left(P\left(v,v_f\right)\right),\mathscr{E}(v_s)\right). \quad (11)$$

Overall, UCT initially prefers nodes that have high similarity with the RNN output $\mathscr{E}_{\mathrm{net}}(\cdot)$ and low visit count $N(\cdot)$, but then asymptotically prefers nodes with high values of $F_S(\cdot)$.

4.5. The Bound of TAG-K Matching. Now, we introduce the upper tree and the lower tree that are variants of MCT to compute the estimate value of upper bound $F_U(\cdot)$ and lower bound $F_L(\cdot)$, respectively. RN-MCTS can stop as soon as Lemma 1 is satisfied, without computing the entire

$M_U(Q, G, u_0)$. The details of the upper tree and lower tree are as follows.

4.5.1. Upper Tree. In the upper tree, given a pattern graph $Q = (V_Q, E_Q, L_Q, \lambda_Q, u_0)$, RN-MCTS relaxes $\lambda_Q = (l, \lambda_t, \lambda_r, \lambda_\rho)$ to be $\lambda_{Q'} = (l', \lambda_t', \lambda_r', \lambda_\rho')$ as follows:

$$\left\{l', \lambda_t', \lambda_r', \lambda_\rho'\right\} = \left\{\max\left(l(e)\right), \min\left(\lambda_t(e)\right), \min\left(\lambda_r(e)\right), \min\left(\lambda_\rho(e)\right)\right\}, \quad e \in E_Q. \tag{12}$$

In addition, in the upper trees, $F_S(\cdot)$ saved in the nodes is replaced by $F_S^U(\cdot)$ and is defined as follows:

$$F_S^U(v, v') = \frac{2\alpha}{\pi}\arctan\left|R_S(v, v')\right| + (1-\alpha)\max_{P_i \in R_{ES}(v,v')} U(P_i). \tag{13}$$

The upper bound $F_U(u_0, v)$ is calculated by equation (7). Thus, comparing with equation (6), because of the relaxation of constraints and the selection of maximal $U(\cdot)$, $F_U(u_0, v) > F(u_0, v)$. In the searching process of upper trees, RNN structure will not be used.

4.5.2. Lower Tree. In the lower tree, the pattern graph and the MCT are not changed, and $F_L(u_0, v)$ is calculated by using equation (6). Hence, with the increase in iterations, $F_L(u_0, v)$ will be close to $F(u_0, v)$. Here, the RNN structure is used to guide the searching process.

4.6. Optimization Strategy. In this section, we introduce 3 *optimization strategies* to reduce the searching space and speed up the searching process.

Strategy 1. Rapid evaluate. Social graphs are typically large, with millions of nodes and billions of edges; hence, RN-MCTS first uses *rapid evaluate* to make the preliminary estimate lower bounds and upper bounds of all candidates and details as below.

Given a designated node u_0 in Q, a candidate v in $C(u_0)$, RN-MCTS builds the upper tree and the lower tree of v and performs once iteration on them, respectively, and then uses the searching results to calculate the initial values of lower bound and upper bound. In the process of rapid evaluate, if the candidate v_c is a descendant in the MCT of v, v_c will be preferentially selected, then RN-MCTS calculates $F_S(v, v_c)$ in the lower tree and $F_S^U(v, v_c)$ in the upper tree; set them as the initial values of $F_L(u_0, v_c)$ and $F_U(u_0, v_c)$, respectively, and then put v_c into the *protection set* PT. The nodes in PT will be not deleted when performing optimization strategies, and v_c will be taken out from PT the next time RN-MCTS searches the MCT of v_c. In addition, $F_L(u_0, v_c)$ and $F_U(u_0, v_c)$ will be updated when v_c is visited again and still in PT. Thus, RN-MCTS can get initial bounds of candidates without visiting all MCTs.

Strategy 2. Optimization at dominating nodes. In the MCTs, there can have multiple paths ending at the same nodes. To obtain a near-optimal solution, the first time the searching path (denoted as P_1) reaches the node v_i, RN-MCTS marks v_i as *visited* and stores the value of $\delta(P_1)$, and assume the parent node of v_i is v_s. In the following iterations, suppose there is another path from root node to v_i(denoted as P_y, assume the parent node of v_i in P_y as v_t). If $\delta(P_1) > \delta(P_y)$, it indicates P_y is better than P_1, then we delete v_i, which is the child node of v_s; otherwise, if $\delta(P_1) < \delta(P_y)$, it indicates P_1 is better than P_y, then we delete v_i that is the child node of v_t.

Strategy 3 (Reduce space). Given a designated node u_0 in Q and two candidates $v_i, v_j \in C(u_0)$, after certain times of iterations, if $F_L(u_0, v_i) > F_U(u_0, v_j)$, it is easy to know v_i may be a better designated node matching than v_j. In order to reduce the search space, RN-MCTS performs optimizations in the following situations:

(i) Situation 1: after certain times of iterations, if $v_j \in C(u_0)$ and $F_U(u_0, v_j) < \sum_{v_i \in C(u_0)}(F_L(u_0, v_i))/|C(u_0)|$, it indicates v_j has a lower probability to be the top-K matches of u_0, then we delete v_j in $C(u_0)$

(ii) Situation 2: for a node $v_j \in C(u_0)$, if all nodes in the upper tree of v_j have been visited and $F_U(u_0, v_j) < F_L^K$, F_L^K is the K-th maximum $F_L(\cdot)$ value of nodes in $C(u_0)$ and it indicates v_j cannot be the top-K matches of u_0, then we delete v_j in $C(u_0)$

4.7. MCTS Details. Each iteration of MCTS can be split up into four steps: *selection*, *expansion*, *simulation*, and *back-propagation*. The four steps of MCTS are discussed below, and the corresponding pseudocode is shown in Algorithm 1.

4.7.1. Probability Function. Before performing the iteration of MCTS, RN-MCTS uses the probability function to randomly select a candidate v_c and the probability of v_c to be selected is defined as follows:

$$P_r(v_c) = \frac{F_H(u_0, v_c)}{\sum_{v_i \in C(u_0)} F_H(u_0, v_i)}. \tag{14}$$

Here, we use a function that is similar to UCT, which combined exploitation with exploration to define the weight of selection:

$$F_H(u_0, v_c) = \mathscr{F}(u_0, v_c) + \theta\sqrt{\frac{\ln\left(\sum_{v_i \in C(u_0)} N(v_i)\right)}{N(v_c)+1}}, \tag{15}$$

and the constant θ controls the level of exploration. $\mathscr{F}(u_0, v_c) = F_U(u_0, v_c) + F_L(u_0, v_c)$.

Step 1 (Selection). In this step, given a candidate $v \in C$ as the root node of the MCT, starting from v, MCTS traverses the current MCT using a tree policy. A tree policy uses an evaluation function (i.e., UCT function) that prioritize nodes with the greatest estimated values. When the searching path

reaches a leaf node, if the leaf node is unvisited and with children yet to be added, MCTS will transition to the expansion step; otherwise, if the leaf node has been visited, then MCTS will transition to the backpropagation step.

Step 2 (Expansion). In the expansion step, for the unvisited leaf node reached in the selection step, MCTS selects all neighbor nodes of the leaf node, and add all nodes that $\delta(\cdot) \leq 1$ into the MCT as child nodes of the leaf node; namely, the searching paths ending at these neighboring nodes are feasible and then initializes $\{P(\cdot), R_{\mathrm{ES}}(\cdot) = \varnothing, R_S(\cdot) = \varnothing, F_S(\cdot) = 0, N(\cdot) = 0\}$ of these child nodes. After that, MCTS transitions to the *simulation* step. If all $\delta(P)$ values of neighboring nodes of v_c are more than 1, it indicates v_c is a leaf node that cannot find relevant paths through v_c, then MCTS will transition to the *backpropagation* step.

Step 3 (Simulation). Since the child nodes of the current node have been added to the MCT, MCTS can continue to traverse the MCT. Therefore, in this step, MCTS performs selection and expansion repeatedly until reaches a visited leaf node and then transitions to the *backpropagation* step.

Step 4 (Backpropagation). Now that the MCTS has reached the visited leaf node, and the rest of the MCT must be updated. Starting from the leaf node, RN-MCTS traverses back to the root node. During the traversal, the statistics stored in each node of the traversal (i.e., $\{R_{\mathrm{ES}}(\cdot), R_S(\cdot), F_S(\cdot)$ or $F_S^U(\cdot), N(\cdot)\}$ are incremented. $N(\cdot) \longleftarrow N(\cdot) + 1$. $R_{\mathrm{ES}}(\cdot), R_S(\cdot), F_S(\cdot)$, or $F_S^U(\cdot)$ are updated as discussed before. In addition, for each node v' in the traversal of backpropagation, we use $\{\mathscr{T}(v, v_c), \mathscr{E}(v'), F_S(v, v')\}$ to train the RNN. Here, v_c is the parent node of v' in the traversal.

We set RN-MCTS performs the above four steps for I iterations. So its time complexity is $O(I \times (\text{selection} + \text{expansion} + \text{simulation} + \text{backpropagation}))$. Let N_c denote the average number of child nodes at each layer of the tree and let D denotes the depth of the search tree. The *selection* step has the time complexity $O(N_c)$, the *expansion* step has $O(N_c)$, and both the *simulation* step and *backpropagation* step have $O(DN_c)$. Therefore, the time complexity of RN-MCTS is $O(IDN_c)$. In addition, we can use graph database to save the graph structure, and in addition, in real-world social networks, not all the pairs of nodes have links. Therefore, we can compress the sparse matrix of a data graph by using the Hybrid format [34] to save large graphs.

5. Experiments

5.1. Experiment Settings

(i) We conduct experiments on five large-scale real-world social graphs available at *snap.stanford.edu*. These datasets have been widely used in the literature for studies of graph pattern matching and social network analysis. The details of these datasets are shown in Table 1.

(ii) We use a popular social network generation tool, SocNetV (socnetv.org, with version 2.2) to generate five query graphs, and the details of these graphs are shown in Table 2, where we randomly select a node from each of the pattern graph as the designated node.

(iii) The average constrains of edges in pattern $(\lambda_t, \lambda_r, \lambda_\rho)$ are set in 0.1, 0.2, 0.3, 0.4, and 0.5 to ensure the high possibility of returning TAG-K designated nodes in a data graph. Otherwise, no or only few answers might be returned by all the algorithms, making it difficult to investigate their performance.

(iv) In each of the datasets, α is set to 0.5; the number of iteration is set to 1000, 2000, 3000, 4000, and 5000; K is set to 5, 10, 15, 20, and 25; and the average maximal bounded path length of the pattern matching is set as 2, 4, 6, 8, and 10, respectively.

(v) In order to avoid the bias in the tree search, we balance the search in both width and depth. Then, in the UCT function and probability function, the exploration parameters of β, γ, and θ are set to 0.5. In the RNN structure, μ is set to 0.5, the embedding of nodes are encoded by DeepWalk [32] with 16 dimensions. We use the ADAM optimization algorithm with a learning rate of 0.005 during training, and we set the mini-batch size to 8.

(vi) In order to deliver fair experimental results to avoid the bias experimental results by a specific setting, we calculate the average ranking function value to illustrate the performance of the proposed method.

5.2. Implementation. In the following experiments, we will compare our RN-MCTS with the basic Monte Carlo Tree Search algorithm without the RNN structure (B-MCTS) and the state-of-art TAG-K method, MC-TAG-K [35]. Since the three algorithms are approximate methods, it is not appropriate to directly compare ranking function values returned by the three algorithms. Therefore, we traverse TAG-K results of the three algorithms and get the true ranking function values, respectively. The performance was investigated by the execution time, the true ranking function values, and the difference between the true ranking function values and the approximate ranking function values.

All MC-TAG-K, B-MCTS, and RN-MCTS algorithms are implemented using Matlab R2019a running on a PC with Intel Core i7-7700K 4.2 GHz CPU, 16 GB RAM, Windows 10 operating system, and MySql 5.7 database. All the experimental results are averaged based on five independent runs.

5.3. Experimental Results and Analysis

5.3.1. Exp-1: Effectiveness. This experiment is to investigate the effectiveness of our RN-MCTS by comparing the average ranking function values of the top-K matches based on different settings of parameters.

Data: CSG $G = (V, E, L_G, \rho)$; query graph $Q = (V_Q, E_Q, L_Q, \lambda_Q, u_0)$; candidate node set $C(u_0)$; number of iterations I;
Perform rapid evaluate to build MCTs of $C(u_0)$;
Use $\{\mathcal{T}(\cdot), \mathcal{E}(\cdot), F_S(\cdot)\}$ of rapid evaluate to train RNN;
for iteration *it* in $[1 \ldots I]$ **do**
  Use probability function to select v from $C(u_0)$; set the current node $v_u = v$;
  In the upper tree and lower tree of v:
  While $\delta(v, v_u) \leq 1$ **do**
    If v_u is unvisited **then**
      Mark v_u visited; add the neighbors v_c of v_u in G that $\delta(P)(v, v_c) \leq 1$ into the MCT as the child nodes of v_u; initial $\{P, R_S, R_{ES}, F_S, N\}$ of these child nodes;
      if v_u.child = $\varnothing$ **then**
        Break;
      end
    end
    if v_u is a leaf node **then**
      Break;
    end
    Select the node v_c with the maximum UCT value; update $v_u \longleftarrow v_c$;
  end
  for each node in $P(v, v_u)$ **do**
    Update $\{P, R_S, R_{ES}, F_S, N\}$; use $\{\mathcal{T}(\cdot), \mathcal{E}(\cdot), F_S(\cdot)\}$ to train RNN;
  end
  Perform optimization strategy;
  if meet the situation of early terminal **then**
    Return top-K matches of u_0;
  end
end
Return top-K matches of u_0.

Algorithm 1: RN-MCTS.

Table 1: The social datasets.

Name	Nodes	Edges	Description
citHepTh	28k	353k	Citation network
Slashdot	77k	905k	Social network
DBLP	317k	1050k	Collaboration network
Twitter	741k	2400k	Social network
YouTube	1729k	3844k	Social network

Table 2: The pattern graphs.

Name	Nodes	Edges
Q_5	5	7
Q_{10}	10	21
Q_{15}	15	34
Q_{20}	20	50
Q_{25}	25	63

Results. Figures 2 and 3 and Table 3 depict the average ranking function values of TAG-K results with different settings of parameters under MC-TAG-K, B-MCTS, and RN-MCTS. From these figures, we can see that the average ranking function values returned by MC-TAG-K are always less than that of B-MCTS and RN-MCTS. Overall, the average true function value delivered by RN-MCTS is 15.92% and 10.04% more than that of MC-TAG-K and B-MCTS, respectively.

Analysis. The experimental results illustrate that (1) the approximate ranking function values returned by MC-TAG-K have a large deviation with the true ranking function values; (2) RN-MCTS and B-MCTS can deliver more accurate approximate ranking function values by using the neural network and MCTS; therefore, RN-MCTS and B-MCTS can get better matching results than MC-TAG-K; (3) with the assistance of neural network and optimal strategy, RN-MCTS can avoid the MCT search via a subgraph that has a lower probability to satisfy the constraints and then outperforms B-MCTS.

5.3.2. Exp-2: Efficiency. This experiment is to investigate the efficiency of our RN-MCTS by comparing the average query processing time of MC-TAG-K, RN-MCTS, and B-MCTS based on different settings of parameters.

Results. Figures 4 and 5 depict the average query processing time of the three algorithms in returning different numbers of designed nodes with different setting of parameters. From these figures, we can see that RN-MCTS and B-MCTS have better efficiency than MC-TAG-K. Statistically, on average, the query processing time of RN-MCTS is 75.30% less than that of MC-TAG-K and 17.30% more than that of B-MCTS.

Analysis. The experimental results illustrate that (1) RN-MCTS can reduce the searching space and avoid to visit all

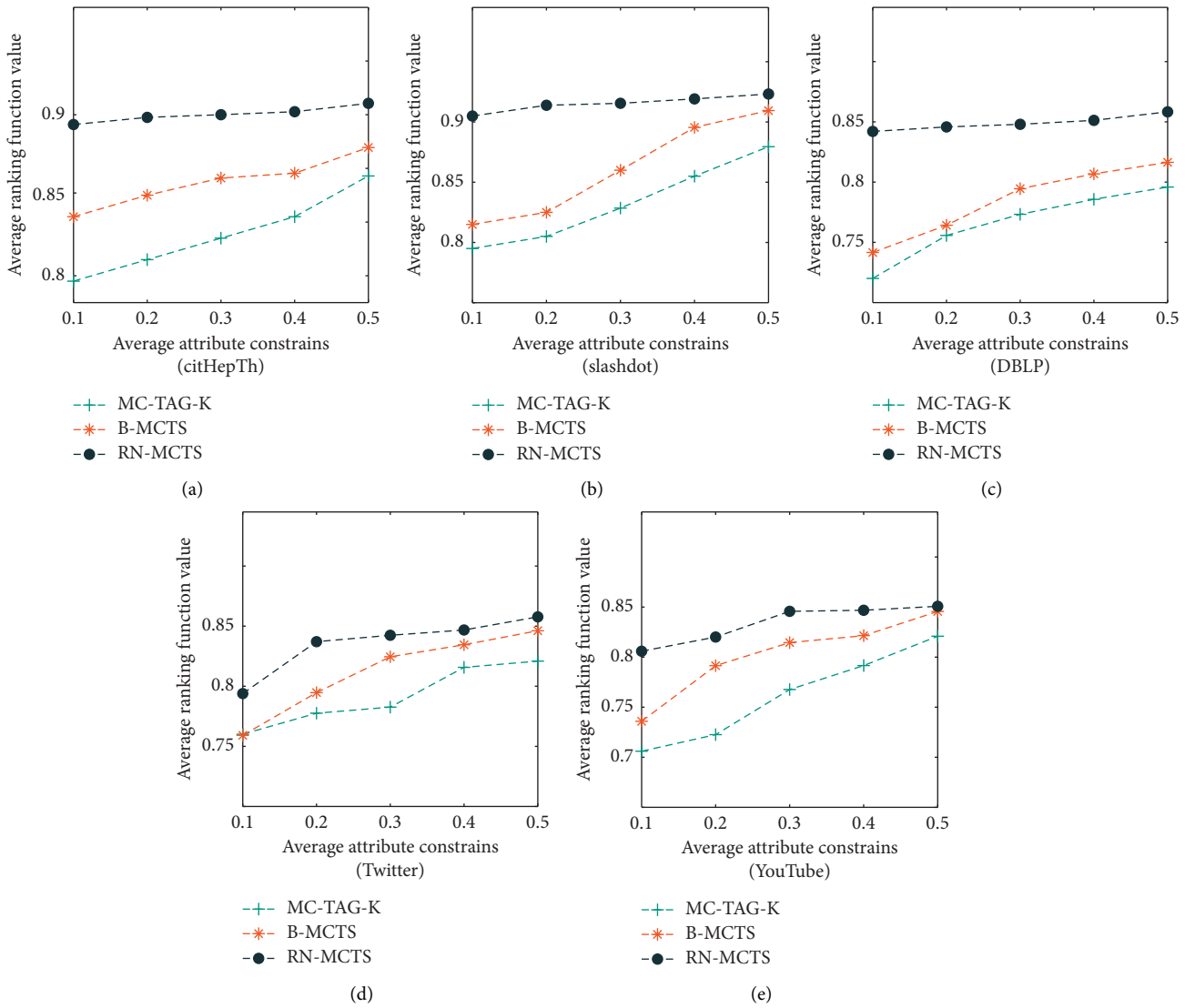

Figure 2: The average ranking function value based on different constrains.

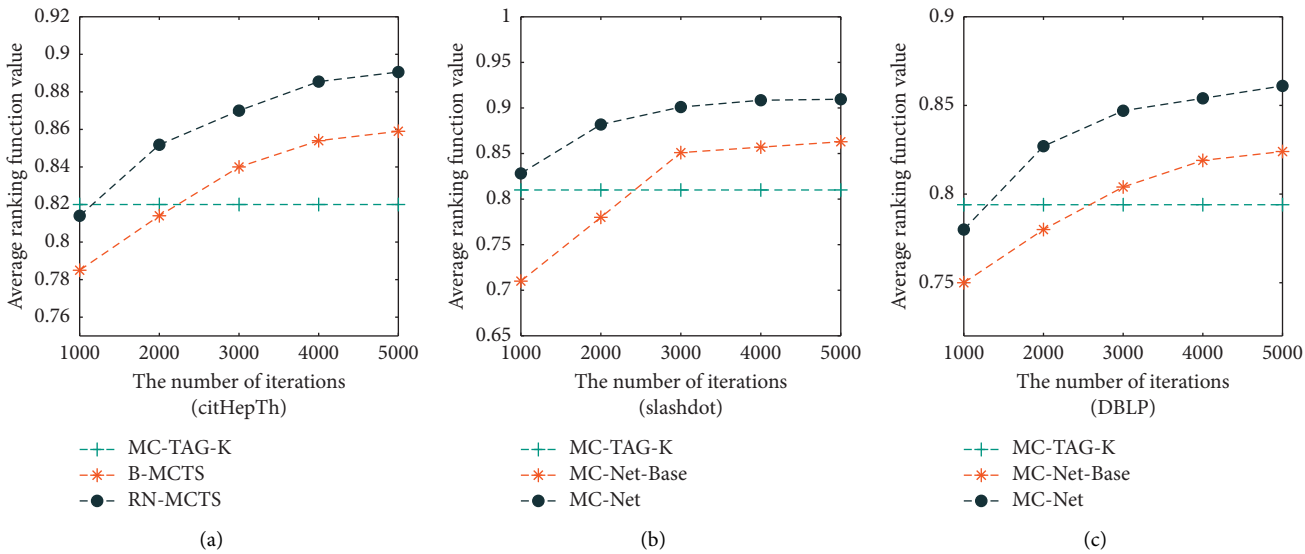

Figure 3: Continued.

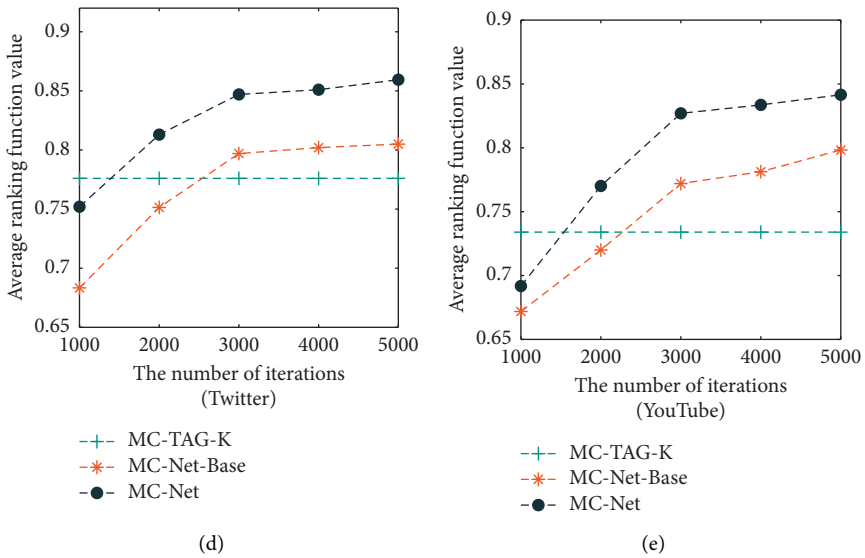

Figure 3: The average ranking function value based on different iterations.

Table 3: The difference between approximate and true ranking function values.

	citHepTh	Slashdot	DBLP	Twitter	YouTube
MC-TAG-K	0.0891	0.0953	0.1275	0.1324	0.1277
B-MCTS	0.0542	0.0465	0.0752	0.0846	0.0861
RN-MCTS	0.0475	0.0412	0.0624	0.0718	0.0724

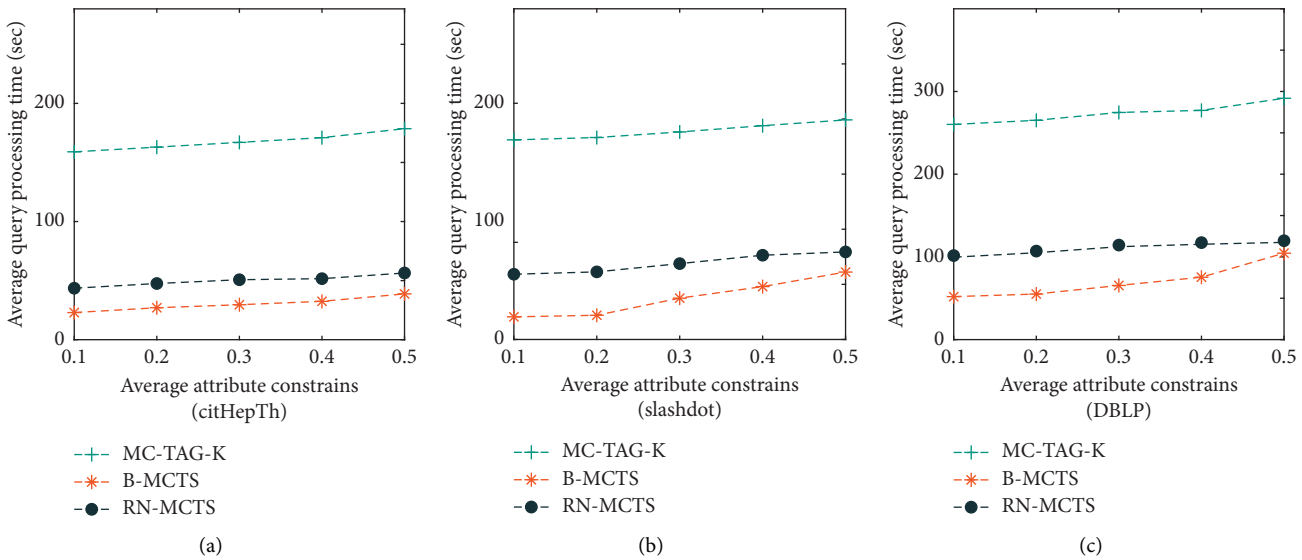

Figure 4: Continued.

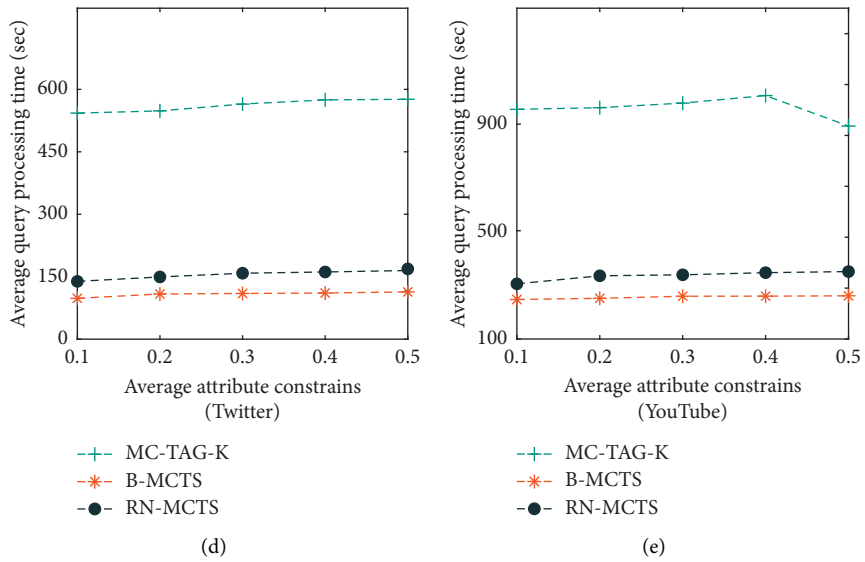

Figure 4: The average query processing time based on different constrains.

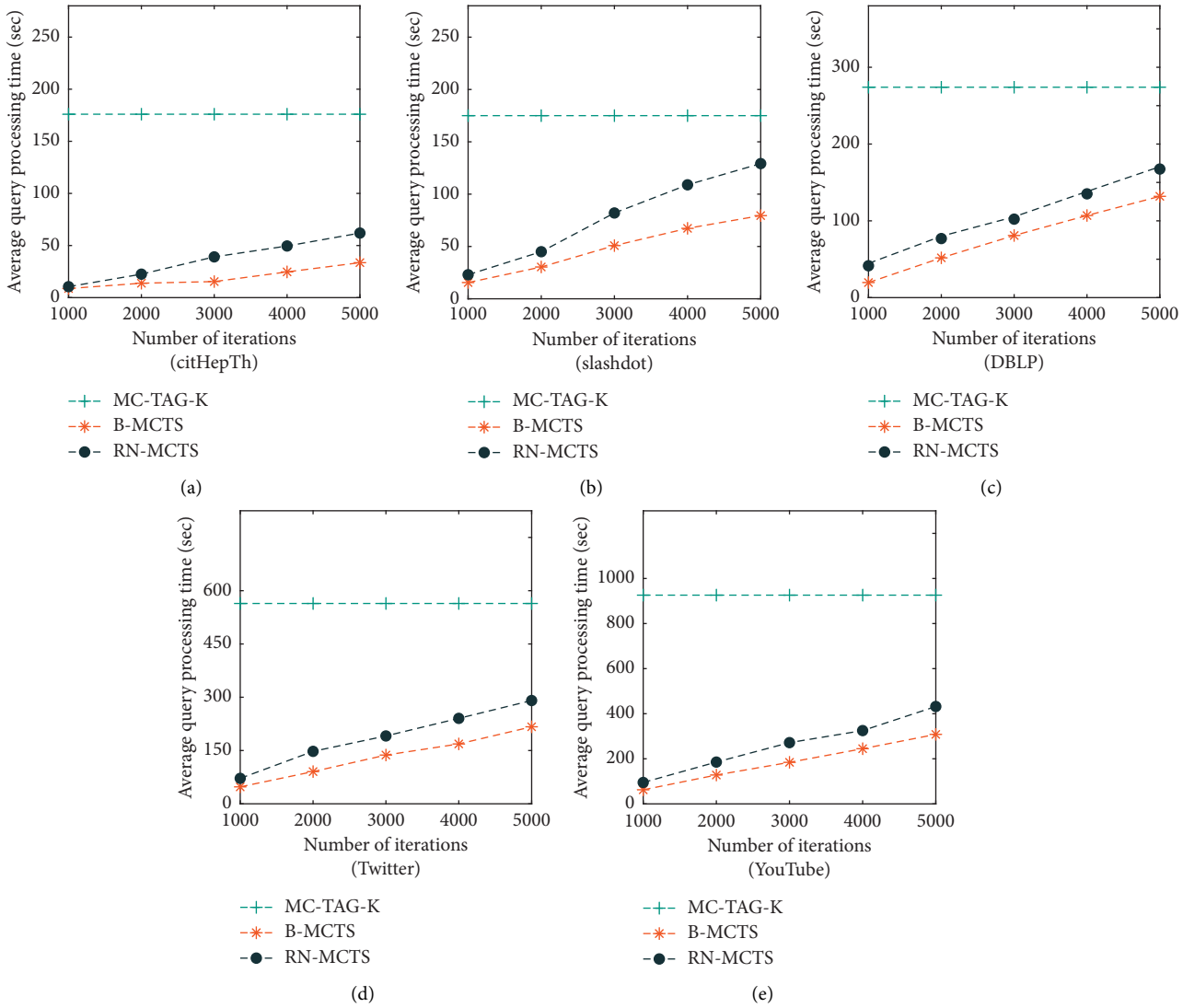

Figure 5: The average query processing time based on different iterations.

nodes and edges in the data graph. Thus, RN-MCTS can greatly save the query processing time than MC-TAG-K; (2) because of the RNN structure, each iteration of RN-MCTS costs more time than B-MCTS.

6. Conclusion and Future Work

In this paper, we have proposed an approximate algorithm RN-MCTS to support a new type context-aware graph pattern-based top-K designated node finding problem that is a corner stone for many social network-based applications. RN-MCTS achieves $O(IDN_c)$ in time cost, and the experiments conducted on five real-world large-scale social graphs have demonstrated the superiority of our proposed approaches in terms of effectiveness and efficiency.

In our future work, we will extend our model to solve the dynamic node matching problem in complex social graphs.

Data Availability

The graph data used to support the findings of this study have been deposited in the Google Drive repository at https://drive.google.com/open?id=1N2-WlMtTR7aRmXw262LtCqxRW2G2VDems.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (NSFC) (Grant nos. 61972069, 61836007, 61832017, 61532018, and 61572336).

References

[1] F. Chevalier, J.-P. Domenger, J. Benois-Pineau, and M. Delest, "Retrieval of objects in video by similarity based on graph matching," *Pattern Recognition Letters*, vol. 28, no. 8, pp. 939–949, 2007.

[2] J. W. Raymond and P. Willett, "Maximum common subgraph isomorphism algorithms for the matching of chemical structures," *Journal of Computer-Aided Molecular Design*, vol. 16, no. 7, pp. 521–533, 2002.

[3] E. Modiano, "Traffic grooming in WDM networks," *IEEE Communications Magazine*, vol. 39, no. 7, pp. 124–129, 2001.

[4] B. Zheng, H. Wang, K. Zheng, H. Su, K. Liu, and S. Shang, "SharkDB: an in-memory column-oriented storage for trajectory analysis," *World Wide Web*, vol. 21, no. 2, pp. 455–485, 2018.

[5] B. Zheng, K. Zheng, C. S. Jensen et al., "Answering why-not group spatial keyword queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 1, pp. 26–39, 2020.

[6] B. Zheng, K. Zheng, P. Scheuermann, X. Zhou, Q. V. H. Nguyen, and C. Li, "Searching activity trajectory with keywords," *World Wide Web*, vol. 22, no. 3, pp. 967–1000, 2019.

[7] M. R. Morris, J. Teevan, and K. Panovich, "What do people ask their social networks, and why?: A survey study of status message Q&A behavior," in *Proceedings of the 28th International Conference on Human factors in Computing Systems—CHI'10*, pp. 1739–1748, Atlanta, GA, USA, April 2010.

[8] R. Schenkel, T. Crecelius, M. Kacimi et al., "Efficient top-k querying over social-tagging networks," in *Proceedings of the SIGIR*, pp. 523–530, Singapore, July 2008.

[9] S. Cohen, B. Kimelfeld, G. Koutrika, and V. Jan, "On principles of egocentric person search in social networks," in *Proceedings of the Workshop on VLDS*, pp. 3–6, Seattle, WA, USA, 2011.

[10] W. Fan, X. Wang, and Y. Wu, "Diversified top-k graph pattern matching," *Proceedings of the VLDB Endowment*, vol. 6, no. 13, pp. 1510–1521, 2013.

[11] R. Milano, R. Baggio, and R. Piattelli, *The Effects of Online Social Media on Tourism Websites*, Springer, New York, NY, USA, 2011.

[12] G. Liu, Y. Wang, and M. A. Orgun, "Optimal social trust path selection in complex social networks," in *Proceedings of the AAAI'10*, pp. 1391–1398, Atlanta, GA, USA, July 2010.

[13] P. Berger and T. Luckmann, *The Social Construction of Reality: A Treatise in the Sociology of Knowledge*, Anchor Books, New York, NY, USA, 1966.

[14] G. Liu, Y. Wang, and M. A. Orgun, "Social context-aware trust network discovery in complex contextual social networks," in *Proceedings of the AAAI*, vol. 12, pp. 101–107, Toronto, Canada, 2012.

[15] G. Liu, K. Zheng, Y. Wang et al., "Multi-constrained graph pattern matching in large-scale contextual social graphs," in *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, ICDE*, pp. 351–362, Seoul, South Korea, June 2015.

[16] X. Ding, J. Jia, J. Li, J. Liu, and H. Jini, "Top-k similarity matching in large graphs with attributes," in *Database Systems for Advanced Applications*, pp. 156–170, Springer, Cham, Switzerland, 2014.

[17] G. Liu, F. Zhu, K. Zheng et al., "TOSI: a trust-oriented social influence evaluation method in contextual social networks," *Neurocomputing*, vol. 210, pp. 130–140, 2016.

[18] Y. Tian and P. M. Patel, "Tale: a tool for approximate large graph matching," in *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pp. 963–972, Cancun, Mexico, April 2008.

[19] J. Zhu, W. Jiang, A. Liu, G. Liu, and L. Zhao, "Time-dependent popular routes based trajectory outlier detection," in *Web Information Systems Engineering*, pp. 16–30, Springer, Cham, Switzerland, 2015.

[20] B. Kartal, E. Nunes, J. Godoy, and M. Gini, "Monte-Carlo tree search for multi-robot task allocation," in *Proceedings of the AAAI*, pp. 4222-4223, Phoenix, AZ, USA, February 2016.

[21] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Computer and Games*, Springer, Berlin, Germany, 2006.

[22] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Lecture Notes in Computer Science*, pp. 282–293, Springer, Berlin, Germany, 2006.

[23] W. L. Hamilton and R. Ying, "Representation learning on graphs: methods and applications," *IEEE Data Engineering Bulletin*, vol. 40, no. 3, pp. 52–74, 2017.

[24] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM*, vol. 23, no. 1, pp. 31–42, 1976.

[25] Y. Zhu, L. Qin, J. X. Yu, and H. Cheng, "Finding top-k similar graphs in graph databases," in *Proceedings of the 15th International Conference on Extending Database Technology—EDBT'12*, pp. 456–467, Berlin, Germany, March 2012.

[26] W. Fani, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu, "Graph pattern matching: from intractable to polynomial time," *VLDB*, vol. 3, no. 1-2, pp. 264–275, 2010.

[27] M. Rauch, H. Thomas, and K. Peter, "Computing simulations on finite and infinite graphs," in *Proceedings of the IEEE 36th Annual Foundations of Computer Science*, pp. 453–462, Milwaukee, WI, USA, October 1995.

[28] L. Chang, X. Lin, W. Zhang, J. X. Yu, Y. Zhang, and L. Qin, "Optimal enumeration," *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 533–544, 2015.

[29] Y. Zhao, K. Zheng, Y. Li, H. Su, J. Liu, and X. Zhou, "Destination-aware task assignment in spatial crowdsourcing: a worker decomposition approach," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[30] K. Zheng, Y. Zhao, D. Lian, B. Zheng, G. Liu, and X. Zhou, "Reference-based framework for spatio-temporal trajectory compression and query processing," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[31] G. Liu, Q. Shi, K. Zheng, A. Liu, Z. Li, and X. Zhou, "An efficient method for top-k graph based node matching," *World Wide Web*, vol. 22, no. 3, pp. 945–966, 2019.

[32] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining—KDD'14*, pp. 701–710, New York, NY, USA, August 2014.

[33] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[34] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–11, Portland, OR, USA, November 2009.

[35] G. Liu, Q. Shi, K. Zheng, Z. Li, A. Liu, and J. Xu, "Context-aware graph pattern based top-k designated nodes finding in social graphs," *World Wide Web*, vol. 22, no. 2, pp. 751–770, 2019.