Distributed and Parallel Databases
https://doi.org/10.1007/s10619-023-07432-2

Scalable top-*k* query on information networks with hierarchical inheritance relations

Fubao Wu[1] · **Lixin Gao**[1]

Accepted: 21 April 2023

Abstract

Graph query, pattern mining and knowledge discovery become challenging on large-scale heterogeneous information networks (HINs). State-of-the-art techniques involving path propagation mainly focus on the inference of node labels, and neighborhood structures. However, entity links in the real world also contain rich hierarchical inheritance relations. For example, the vulnerability of a product version is likely to be inherited from its older versions. Taking advantage of the hierarchical inheritances can potentially improve the quality of query results. Motivated by this, we explore hierarchical inheritance relations between entities and formulate the problem of graph query on HINs with hierarchical inheritance relations. We propose a graph query search algorithm by decomposing the original query graph into multiple star queries and applying a star query algorithm to each star query. Candidates from each star query result are then constructed for the final top-*k* query answer to the original query. To efficiently obtain the graph query result from a large-scale HIN, we design a bound-based pruning technique by using the uniform cost search to prune the search spaces. We implement our algorithm in Spark GraphX to test the effectiveness and efficiency on synthetic and real-world datasets. Compared with two state-of-the-art graph query algorithms, our algorithm can effectively obtain more accurate results and competitive performance.

Keywords Heterogeneous information network · Graph query · Hierarchical inheritance relations

✉ Fubao Wu
fubaowu@umass.edu

Lixin Gao
lgao@ecs.umass.edu

[1] Department of Electrical and Computer Engineering, University of Massachusetts Amherst, Amherst, USA

Published online: 03 June 2023

1 Introduction

Many real-world systems, such as enterprise networks, social networks, and biological networks, can be modeled as heterogeneous information networks (HINs) [1–4]. A HIN contains multiple types of objects and relations providing rich semantic queries, knowledge discoveries, information fusions, recommendations, and predictions. Graph query, as an important technique for solving these tasks, has been extensively explored recently. It mainly explores subgraph isomorphism algorithms to get an exact match [5, 6], and also develops subgraph matching algorithms to do an inexact/approximate match as the potential query answers [7, 8]. Current research on graph query/matching mainly focuses on two dimensions. The first dimension is the unary node-to-node properties mapping. The second dimension is edge-to-edge/path similarities. Jin et al. [8, 9] consider node types and closest path propagation to get scores of query answers. Some works [7, 8, 10, 11] consider similar nodes' labels and their neighbors to learn the path propagation to get ranked answers.

However, knowledge representation has hierarchical structures in the real-world system. Long et al. [12] state that the knowledge structure representation can be inherited with upward and downward inheritances. Clauset et al. [13] show that the existing knowledge of hierarchical structure can be used to predict missing connections. In addition, Jiang et al. [14] construct the hierarchical structures of entities for the large freebase knowledge base system based on real-world entities and relations. One visible example in an enterprise's product databases is that product vulnerabilities can be inherited from or passed down to different product versions. While measuring the similarity of objects for graph matching, hierarchical inheritance relations can also play an important role in the answer ranking. The quality of query answers is also greatly affected by hierarchical inheritance relations. Therefore, we consider the power of hierarchical inheritances whereby a subclass inherits the properties and constraints of its parents, and more meaningful and accurate query answers are expected to be obtained.

Taking an example of an information network with a hierarchical structure, we consider a schema of an enterprise's product information network shown in Fig. 1a. Every node represents a type of entity at the schema level. The product type is connected by four property types: site, workgroup, technology, and vulnerability. Product entities have hierarchical connections with different versions of the products shown in Fig. 1b. Some properties are inherited among different versions of the products, such as vulnerability and technology properties (in red bold lines in Fig. 1a).

Given the information network schema with inherited relations, we show a user query example here. Assume a user wants to find the top-5 related products affected with a given vulnerability V_1 (Cisco WebEx meetings server information disclosure vulnerability) and employed with a given technology T_1 (voice-communications manager additional apps and plugins), which is constructed as a user query graph shown in Fig. 1c. Figure 1d shows the top-5 subgraph answers of the query in this answer graph. The given V_1 and T_1 nodes in the user query graph are

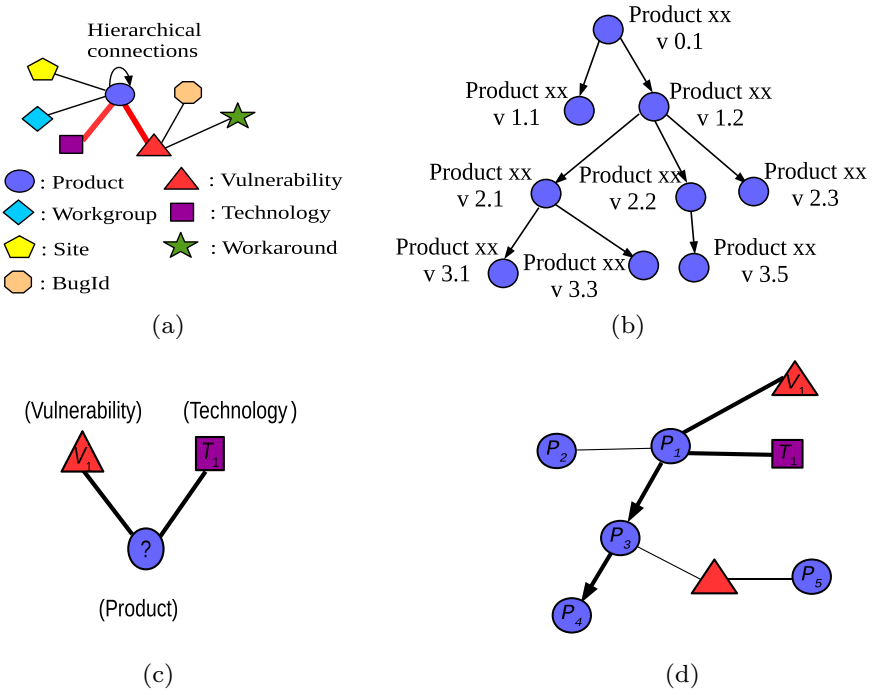


Fig. 1 A product information network schema and a query example. **a** A graph schema of a product network. **b** An example of a hierarchical connection structure for product versions. **c** A user query graph of top products. **d** An answer graph with top-5 products

exactly matched with the V_1 and T_1 nodes, respectively, in the answer graph, and there are 5 product nodes that are potential answers to the query of the product node.

For general methods, if we consider the closest node types and shortest distance to measure the similarities of answers for matching, we obtain the following ranking order of answer scores, P_1 (Cisco WebEx meetings server versions 0.1.0), P_2 (Cisco WebEx meetings server versions 0.2.0), P_3 (Cisco WebEx meetings server versions 1.1), P_4 (Cisco WebEx meetings server versions 2.1) and P_5 (Cisco Jabber for Windows), that is, the ranking order of answer scores is $s(P_1) > s(P_2) = s(P_3) > s(P_4) > s(P_5)$. However, the vulnerability property can be inherited from different prior versions of products. Here P_1 is the prior (parent) version of P_3, and P_3 is the parent version of $P4$ as the arrows indicate. Hence, P_1's vulnerabilities can pass down to the product P_3 or P_4, and P_4's vulnerability can come from the upper P_3 or P_1. With the hierarchical inheritances, the answer scores can be obtained with a more accurate ranking order $s(P_1) \approx s(P_3) \approx s(P_4) > s(P_2) > s(P_5)$, which is very important for engineers' troubleshooting and customers' queries.

Due to the complexity and heterogeneousness of large networks, designing an effective and efficient algorithm with additional hierarchical features

is challenging. In this paper, we conquer this problem by modeling graph queries with a new matching score function with hierarchical inheritance relations for effective answers, and by proposing a bound-based technique for an efficient query. The main contributions are as follows:

- We formulate the graph query problem with hierarchical inheritance relations to improve the query quality.
- We propose a new graph query algorithm based on uniform cost search in the context of a new matching score function.
- We design a bound-based method to prune the search spaces to efficiently get the top-k best answers.
- We implement our algorithm in the Spark GraphX distributed environment for large-scale networks. Experiments are done to evaluate the effectiveness and efficiency of our matching algorithm.

The rest of this paper is organized as follows. Section 2 describes the problem and formulates graph queries with hierarchical inheritance relations. The proposed algorithm for graph queries and its bound-based pruning technique are presented in Sect. 3. Section 4 discusses the distributed implementation. In Sect. 5, we present the evaluation of our algorithms. The related work and conclusion are shown in Sects. 6 and 7, respectively.

2 Problem formulation

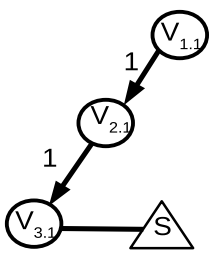
2.1 Data graph, hierarchical inheritance relations, query graph, and matching

Definition 1 (Data Graph) We consider a HIN that contains hierarchical inheritance relations among nodes as a hierarchical heterogeneous information network (HHIN). A HHIN is modeled as a partially directed, acyclic, labeled data graph $G(V, E, L_v, H_e)$ with a node set V, edge set E, node label set L_v and hierarchical inheritance relations H_e with directions, where (1) each node $v \in V$ represents an entity in G, (2) each edge $e \in E$ represents the relationship between two entities, and each edge weight is considered to be 1. Only an edge between two hierarchical entities has a direction. (3) each node v has the label information L_v, including at least a node type and a keyword description, (4) for hierarchical entities, each edge $e \in H_e$ between them indicates a hierarchical inheritance relation. Each edge weight h_e between two hierarchical entities as the hierarchical level distance is defined as $|h_e| = 1$.

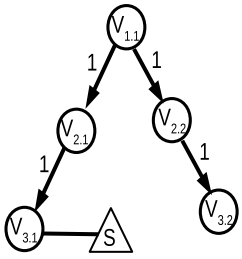
Definition 2 (Hierarchical Inheritance Relations) There exist upward and downward hierarchical inheritance relations in G. We call a node with label information that is inherited among other hierarchical entities an "attaching" node, such as a vulnerability node in Fig. 1a. Its label information could be inherited among product nodes. A node with a node type that has hierarchical levels is called an "inherited" node, such

as a product node in Fig. 1a. If an attaching node's label information passes down to its inherited node's lower level entity, we call it downward inheritance. Conversely, if an attaching node's label information can pass up to its inherited entity's higher level entity, it is called upward inheritance. The attaching node and one of its inherited nodes are formed as a "property inheritance pair". For example, the vulnerability's label information in the vulnerability entity can be downward or upward inherited from product entities in higher or lower levels as shown in the red bold line in Fig. 1a. The label information of other nodes, such as workgroup and site, is not inherited among product nodes as shown in the black line in Fig. 1a.

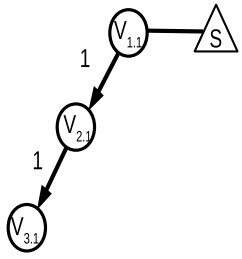
Given an hierarchical edge $h(u_1, u_2)$ between node u_1 and u_2, the hierarchical level difference is 1 for upward inheritance when u_1 is in the higher level than u_2, -1 for downward inheritance when u_1 is in the lower level than u_2, and 0 for the same hierarchical level or non-hierarchical relations. Considering hierarchical inheritance cases shown in Fig. 2, given a node pair in the query graph Q, there are possible pairs of nodes $(S, V_{i,j})$ in data graph G, which match with that node pair in Q. S is an "attaching node" and $V_{i,j}$ $(i, j \in 1, 3$ in this example) indicates different



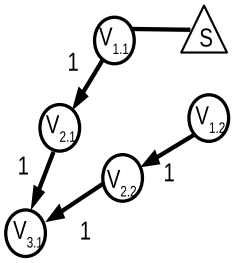
(a) Score $r(S, V_{1.1}) \lessapprox r(S, V_{3.1})$



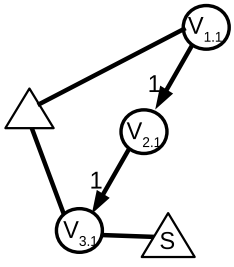
(b) Score $r(S, V_{3.2}) \gtrapprox r(S, V_{1.1})$



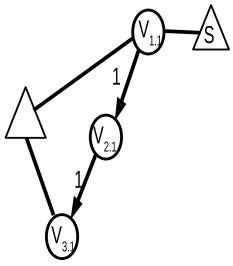
(c) Score $r(S, V_{3.1}) \lessapprox r(S, V_{1.1})$



(d) Score $r(S, V_{1.2}) \gtrapprox r(S, V_{3.1})$



(e) Maximum hierarchical difference: $max(h(S, V_{1.1})) = -2$



(f) Maximum hierarchical difference: $max(h(S, V_{3.1})) = 2$

Fig. 2 Hierarchical inheritance cases and scores

"inherited" candidate nodes. There are basically these following cases if we only consider the matching based on hierarchical inheritance relations.

1. When S inherits upward from $V_{3.1}$ to $V_{1.1}$, the node similarity score r of S to $V_{1.1}$ is a little smaller than the score of S to $V_{3.1}$, that is, $r(S, V_{1.1}) \lessapprox r(S, V_{3.1})$ shown in Fig. 2a.
2. When S inherits upward from $V_{3.1}$ to $V_{1.1}$ and downward from $V_{1.1}$ to $V_{3.2}$, it is expected that the node similarity score $r(S, V_{3.2}) \gtrapprox r(S, V_{1.1})$ shown in Fig. 2b.
3. When S inherits downward from $V_{1.1}$ to $V_{3.1}$, it is expected that the node similarity score $r(S, V_{3.1}) \lessapprox r(S, V_{1.1})$ shown in Fig. 2c.
4. When S inherits downward from $V_{1.1}$ to $V_{3.1}$ and upward from $V_{3.1}$ to $V_{1.2}$, it is expected that the node similarity score $r(S, V_{1.2}) \gtrapprox r(S, V_{3.1})$ shown in Fig. 2d.
5. When S inherits upward from $V_{1.1}$ to $V_{3.1}$, and there are multiple shortest paths from S to $V_{1.1}$ with different hierarchical level differences, the maximum hierarchical difference is defined as: $max(h(S, V_{1.1})) = -2$ shown in Fig. 2e.
6. When S inherits downward from $V_{1.1}$ to $V_{3.1}$, and there are multiple shortest paths from S to $V_{3.1}$ with different hierarchical level differences, the maximum hierarchical difference is defined as: $max(h(S, V_{3.1})) = 2$ shown in Fig. 2f.

Definition 3 (Query Graph) A query graph $Q(V_Q, E_Q, L_v)$ is modeled as an undirected and labeled graph. V_Q contains a set of specific nodes V_Q^S and a set of query nodes V_Q^U with types τ_Q^U, which are provided by users. A specific node is defined as an instantiated node in Q that has a fixed node type and node label information, and it is also matched to a node in G. A query node is defined as a node in Q that only its node type is given, and we want to find its matched nodes in G. According to one classification category based on query node number in Q, if the query node number $|V_Q^U| = 1$, we denote the query graph Q as a star query graph. If the query node number $|V_Q^U| > 1$, Q is called a general (non-star) query graph. According to another classification category based on hierarchical inheritance relations, if every one of the specific nodes in V_Q^S can form a property inheritance pair with its query node in V_Q^U, we call Q a hierarchical query graph. If there exist no property inheritance pairs, we call Q a non-hierarchical query graph. Otherwise, it is called a mixed hierarchical query graph. For example, Fig. 1c shows a hierarchical star query graph where node V_1 and node T_1 comprise specific nodes, and the node marked with the "?" in the product type represents a query node.

Given a query graph Q and a data graph G, we need to map each query node to a data node. This transfers to a subgraph matching problem. We denote as M an already matched subgraph in G to Q. Then a subgraph matching is a many/one-to-one mapping function ϕ: $V_Q \rightarrow V$, such that, for each query node $v \in V_Q$, $\phi(v) \in M$. The problem here is to find such top-k potential mapping functions given a query graph Q and a data graph G.

2.2 Matching score

If nodes are close in a query graph, their mapping nodes in a data graph are also close based on node neighbors and hierarchical inheritance relations. Given a query graph Q containing a node pair $(u, v) \in V_Q$ that is connected, a matched subgraph M in G has mapped nodes $(\phi(u), \phi(v))$.

To measure how close is a query node in Q to its mapping node in G, we define "node closeness score" based on whether hierarchical inheritances exist in Q.

(1) When u and v in Q do not form a property inheritance pair, the closeness score of $(\phi(u), \phi(v))$ is defined similar to [9]. Between two nodes, it considers their shortest distance in the graph with an exponential function for a monotonically decreasing relationship. The shorter the distance, the higher the closeness score.

$$r(\phi(u), \phi(v)) = \begin{cases} 1 & \text{if } \phi(u) = \phi(v) \\ \alpha^{l(\phi(u),\phi(v))} & \text{otherwise} \end{cases} \tag{1}$$

where $l(\phi(u), \phi(v)$ is the shortest distance from $\phi(u)$ to $\phi(v)$. α is a constant propagation factor in [0, 1] that controls the decreasing rate of node closeness.

(2) When u and v can form a property inheritance pair, we consider both shortest distance and hierarchical inheritance relations. The motivation for this is that the hierarchical inheritance level also has an important impact on the query results. The closeness score is decreasing with the increasing length of the level, but the inheritance relations have a more positive drive than the short distance. The inheritance level difference will be deducted to some extent from the shortest distance. Therefore, we define the closeness score of $(\phi(u), \phi(v))$ as:

$$r(\phi(u), \phi(v)) = \begin{cases} 1 & \text{if } \phi(u) = \phi(v) \\ \alpha^{l(\phi(u),\phi(v)) - \beta * |h(\phi(u),\phi(v))|} & \text{otherwise} \end{cases} \tag{2}$$

where $l(\phi(u), \phi(v))$ is the shortest distance from $\phi(u)$ to $\phi(v)$. α is a constant propagation factor in [0, 1] that controls the decreasing rate of the node closeness. β is defined as the hierarchical level propagation factor in (0, 1), which indicates the importance of hierarchical level propagation when an attaching node's label information inherits between different hierarchical levels of an inherited node. β is expected to be smaller than α because hierarchical inheritance is more reliable than shortest distances when traversing long hops. $h(\phi(u), \phi(v))$ indicates the hierarchical level difference from $\phi(u)$ to $\phi(v)$. Vice versa, the hierarchical level difference from $\phi(v)$ to $\phi(u)$ is indicated as $h(\phi(v), \phi(u))$, and $h(\phi(v), \phi(u)) = -h(\phi(u), \phi(v))$.

Based on the node closeness score, the matching score of M is defined as the summation of mapping nodes $(\phi(u), \phi(v))$ for all connected edges (u, v) in Q.

$$F(\phi) = \sum_{(u,v) \in E_Q} r(\phi(u), \phi(v)) \tag{3}$$

2.3 Problem statement

Given a query graph Q and a data graph G, we want to find the top-k subgraph answers in G, that is, to find a set of k subgraphs M_k in G, such that for any nodes $\phi(V_Q) \in M_k$ and for all nodes $\phi'(V_Q) \notin M_k$, the matching score $F(\phi) > F(\phi')$. Specific nodes V_Q^S in Q are identified for exactly one-to-one mapping to matched nodes $\phi(V_Q^S)$ in G (we call them anchor nodes V_G^A), which are easy to be found. Therefore, we consider the top-k sets of candidate nodes in M_k for a set of query nodes based on hierarchical inheritance relations and graph structures.

Formally, given a query graph $Q(V_Q, E_Q, L_v)$, the top-k subgraphs $M_k(V', E', L')$ have the following mapping function with Q. For each $v \in V_Q$, there is a one-to-one mapping $\phi(v) \in V'$: $v \rightarrow \phi(v)$ based on the matching score F. Our problem considers exact and approximate matches to output the top-k matching answers, so each edge $e \in E_Q$ does not need to have a one-to-one mapping to the edge $e' \in E'$.

3 Graph query algorithm with hierarchical inheritance relations

It is time-consuming to get all potential subgraphs from a large-scale data graph with a big query graph. Moreover, for a general query graph with multiple query nodes, it is proved to be an NP-hard problem even for subgraph isomorphism [15]. Yang et al. [16] divide a query graph into star queries and then utilize the top-k star-join method, using the similar relational database HRJN [17]. Inspired by the structure of our general query graph with multiple query nodes, we propose a general graph query algorithm comprising three phases as follows:

Phase 1 (Query Decomposition): A general query graph contains some specific nodes and one or more query nodes. Considering the characteristics of our query graph, the decomposing policy of a general query graph is not as complex as the decomposing method considered in [16], as we don't use the join operation for final combinations of star queries. Therefore, a simple and effective policy is to use the number of query nodes as the number of star query graphs. Each query node is the center query node for each star query, and every specific node that is connected to the center query node is a specific node for its star query.

Phase 2 (Star query): We propose to use uniform cost search and bound-based pruning to derive top-k_s candidates for each star query. Selecting the top-k_s candidates for each star query can effectively serve the final top-k candidate results for a general graph query (Sect. 3.1–3.4).

Phase 3 (Candidates selection): We consider the top-k_s star query candidates together and get the optimal edge/path matching scores for query node combinations. Different from the top-k join strategy with a common node in [16], here query node candidates can be 1 or more hops connected in G without a common node for joining. Therefore, graph traversals are needed among these star query node candidates to find the final top-k candidate sets for query nodes. When there are $|V_Q^U|$ query nodes, this involves exponential computations of $|V_Q^U|^{k_s}$, which is highly

expensive if $|V_Q^U|$ and k_s are large. We propose to use a branch and bound technique to greatly reduce search spaces by filtering out unexpected candidate sets (Sect. 3.5.3).

If the input query graph Q is a star query graph, then we only do phase 2 (star query) to get the answer. If the input query graph Q is a general query graph, it will involve the three phases. As query decomposition is easy to accomplish, we will mainly discuss the star query algorithm and candidate selection for the general query graph algorithm.

3.1 Matching score for star query

Given a star query graph Q with a set of specific nodes V_Q^S and a query node v, specific nodes V_Q^S have mapped to anchor nodes V_G^A in G, so we only need to find the top-k mapping nodes $\phi(v)$ for v. We traverse from every anchor node $\phi(v^s)$ in V_G^A as one source with uniform cost search to all other nodes in the data graph, and obtain their node closeness score to the source. We could calculate the matching score of one $\phi(v)$ denoted as $S(\phi(v))$, which is based on the aggregated results of node closeness scores from all the nodes in $\phi(V_Q^S)$:

$$S(\phi(v)) = \sum_{v^s \in V_Q^S} r(\phi(v^s), \phi(v)) \tag{4}$$

3.2 Bound-based pruning for star query

For each different anchor node, there is a propagation path to each candidate node in G. It is time-consuming to do all node traversals if G is very large. We use the bound-based pruning technique to effectively reduce the search space for star queries. We trace the lower bound in the top-k answers and infer the upper bound of unseen nodes to effectively filter these nodes while traversing.

3.2.1 Bounds of matching score

In the top-k answer list of query nodes, every node is maintained with an upper bound of matching score and a lower bound of matching score for a query node. We refine the upper bound $\overline{S}(\phi(v))$ and lower bounds $\underline{S}(\phi(v))$.

$$\overline{S}(u) = \sum_{v^s \in V_Q^S} \overline{r}(\phi(v^s), \phi(v)) \tag{5}$$

$$\underline{S}(u) = \sum_{v^s \in V_Q^S} \underline{r}(\phi(v^s), \phi(v)) \tag{6}$$

The matching score bound depends on the upper bound of node closeness score $\overline{r}$ and the lower bound of closeness score $\underline{r}$. Next, we show how to get these bounds of node closeness score.

3.2.2 Bounds of node closeness score

The lower bound and upper bound are obtained online while the graph traversal is running. We show the lower and upper bound refinement in the different iterations of graph traversal. We denote t as the iteration number of uniform cost search from an anchor node s to a candidate node u.

(1) The initial bounds are set as $\overline{r}^0(s,u) = 1 \quad and \quad \underline{r}^0(s,u) = 0$. (2) In each of the next iterations, every node u is updated with its lower bound using the information from its previous iteration result when it is not visited yet. The lower bound is computed as follows:

$$\underline{r}^t(s,u) = \begin{cases} \underline{r}^{t-1}(s,u) & \underline{r}^{t-1}(s,u) > 0 \\ \alpha^{1-\beta\cdot|h(u_{prev},u)|} \cdot \underline{r}^{t-1}(s,u_{prev}) & otherwise \end{cases} \tag{7}$$

The upper bound in iteration t is computed as follows:

$$\overline{r}^t(s,u) = \begin{cases} \underline{r}^t(s,u) & \underline{r}^t(s,u) > 0 \\ \alpha^{t-\beta\cdot|h(s,u)|} & otherwise \end{cases} \tag{8}$$

where u_{prev} is the parent node of u when traversing from s along a path to u.

3.3 Top-*k* selection with bounds

How to effectively update potential candidate results and select the final top-k results during iterations is crucial for computation performance. Here we use a top-k selection policy based on the upper and lower bounds of matching scores referred to as the top-k emergence test in [18]. We maintain the top-k candidate results in a priority queue P. Each candidate node u contains its lower bound $\overline{S}_G(u)$, and upper bound $\underline{S}_G(u)$. We define $\underline{S}_{kth}$ as the smallest lower bound in P. The process for selecting and updating P during the iterations is shown as follows:

(1) Find the top-k potential answer nodes and put in P. **(2)** Calculate the kth smallest lower bounds $\underline{S}_{kth}$ in P. **(3)** If the upper bound $\overline{S}_G(u)$ of an incoming node n is less than the $\underline{S}_{kth}$, we prune the node u and the nodes with bigger distance than u from the starting source. These nodes' matching scores are lower than any node's matching score in P, so they are not qualified for top-k final results. **(5)** Continue the previous steps until the convergence condition is reached, which is shown in Sect. 3.6.

3.4 Star query algorithm

As discussed before, the star query is to find the match for the star query graph in the large data graph. The star query graph has a uniform structure, in which a small graph Q has a set of specific nodes V_Q^S and a query node, and specific nodes V_Q^S con-

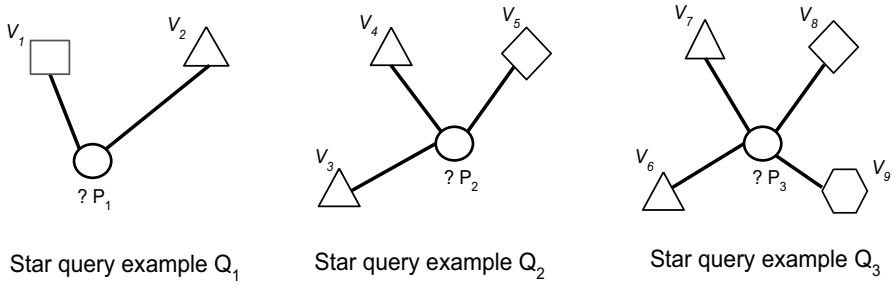


Fig. 3 Star query graph examples

nects to the query node. Figure 3 shows 3 examples of star queries, and each has a query node p that connects to its own different number of specific nodes V_i ($i \in$).

According to the proposed star query matching score and bounding-based pruning, we show our star query with hierarchical inheritance relation algorithm (SQH) in Algorithm 1. First, we obtain anchor nodes $\phi(V_Q^S)$ in G for each V_Q^S in Q, which are specific one-to-one mappings in G (in Line 1). Then we aggregate node messages to do propagation simultaneously from each anchor node with the uniform cost search (in Line 8). The search cost of each node in the uniform cost search is indicated by the inverse of its matching score here. In each iteration of propagation, the candidate node closeness and matching score, lower bounds, and upper bounds are updated (in Line 9–10). Candidate nodes and the queue are continuously updated (in Line 11–15). The specific top-k selection and update are shown (in Line 17–28). Iterations continue until we find the final top-k candidate result. The worst time complexity is $O(|V| * |V_Q^S|)$, where $|V|$ is the node number of G. With the pruning of potential unmatched nodes, the average time complexity is reduced to $O(M * |V_Q^S|)$, where M is the number of visited nodes with type τ and $M \ll |V|$.

Algorithm 1 Top-k star query (SQH)

Require: Data graph $G(V, E, L_v, H_e)$, Query Graph $Q(V_Q^S, \tau)$, Top-k value
Ensure: Top-k match set P_k

Get anchor nodes set $\phi(V_Q^S)$ for V_Q^S
Initialize empty match set P_k (size k)
Initialize node closeness score $r(s,u), \overline{r}(s,u)$ and $\underline{r}(s,u)$
Initialize matching score $(S_G(u), \overline{S}_G(u), \underline{S}_G(u))$
Initialize $L \leftarrow \{v | type(v) = \tau \ \& \ v \in V\}$
$t \leftarrow 0$
while L not empty and message exists **do**
    Aggregate node u from each anchor nodes with uniform cost search
    Update $(r(s,u), \overline{r}(s,u), \underline{r}(s,u))$ by equation 7 and 8
    Update $(S_G(u), \overline{S}_G(u), \underline{S}_G(u))$ by equations 5 and 6
    if $\overline{S}_G(u) - \underline{S}_G(u)) <= 0$ **then**
        $L \leftarrow L - u$
    else
        $P_k = P_k + u$
    $P_k, L \leftarrow$ **TOPKUPDATEBOUNDPRUNE**
    $t \leftarrow t + 1$
procedure TOPKUPDATEBOUNDPRUNE
    $\underline{S}_{kth} \leftarrow k_{th}$ smallest $\underline{S}_G(u)$ for $u \in P_k$
    node n, $S_{kth} \leftarrow k_{th}$ smallest $S_G(u)$ for $u \in P_k$
    for all $u \in P_k$ **do**
        if $size(P_k) < k$ **then**
            $P_k \leftarrow P_k + u$
        else if $\overline{S}_G(u) > \underline{S}_{kth}(n)$ and $S_G(u) > S_{kth}$ **then**
            $P_k \leftarrow P_k - n$
            $P_k \leftarrow P_k + u$
        else if $\overline{S}_G(u) < \underline{S}_{kth}(n)$ **then**
            $L \leftarrow L - u$
    return P_k, L

3.5 General graph query algorithm

The general graph query problem involves three phases described in the earlier part of Sect. 3: decomposing query (phase 1), star query (phase 2), and candidate selection (phase 3). The previous 2 phases have been described before. For phase 3, how to effectively and efficiently select the top matching candidate sets from star query results involves effective candidate selections. We propose to find the top matching scores of query node combinations by propagations. First, we obtain the matching score of query nodes for the general graph query.

3.5.1 Matching score of query nodes

Based on the calculation of the matching score for star queries in Sect. 3.1, we obtain the matching score for a set of query nodes V_Q^U as:

$$F_G(V_Q^U) = \sum_{v \in V_Q^U} S_G(\phi(v)) + \sum_{(v_i, v_j) \in E(V_Q^U)} E_G(\phi(v_i), \phi(v_j)) \tag{9}$$

The summation is composed of two parts. The first part is the summation of matching scores of decomposed star queries. The second part is the summation of matching scores of edges/paths among the candidates of query nodes.

3.5.2 Algorithm flow

We show our general query with hierarchical inheritance relations (GQH) in Algorithm 2. Phase 1 for decomposing query is shown in Line 3. Phase 2 for star query is shown in Line 5–7. The candidate selection (in Line 8–12) continues propagating by uniform cost search from top candidates nodes and pruning with branch and bound until the top-k candidate node set is found. The worst time complexity is $O(|V| * |V_Q^S| + |V| * |V_Q^U|^{k_s})$, where $|V_Q^S|$ is the maximum number of specific nodes for each query node in a query graph, and k_s is the number of top-k_s candidate results from each star query result. In our experiment, $k_s \in [k, 2k]$ is a good trade-off for efficiency and effectiveness. $|V|$ is the number of nodes in G. With the pruning of potential unmatched nodes for phase 2 and phase 3, the worst time complexity is reduced to $O(M * |V_Q^S| + N * |V_Q^U|^{k_s})$, where M and N are the numbers of visited nodes with type τ for phase 2 and phase 3, respectively.

Algorithm 2 Top-k general query (GQH)

Require: Data graph $G(V, E, L_v, H_e)$, Query Graph $Q(V_Q^S, \tau)$, Top-k value
Ensure: Top-k matched candidate sets Mt
1: Initialize top-k matched candidate sets $Mt \leftarrow \phi$ (size k)
2: Initialize star query result list $stResList \leftarrow \phi$
3: Star query graph set $StarGraphSet \leftarrow$ Query Graph Q
4: $k_s \leftarrow [k, 2k]$
5: **for all** $starGraph \in StarGraphSet$ **do**
6: Top-k_s candidate result $starCand \leftarrow$ SQH (G, $starCand$, k) with Algorithm 1
7: $stResList \leftarrow stResList + starCand$
8: $i \leftarrow 0$
9: **while** $i < len(stResList) - 1$ **do**
10: Traverse from $stResList[i] \rightarrow stResList[i + 1]$
11: Pruning nodes and path with bounds until top-k candidate node sets are found
12: $i \leftarrow i + 1$

3.5.3 Candidate selections with branch and bound pruning

The output of a star query graph is top-k_s candidate nodes for each query node. The problem is how to efficiently connect the candidate nodes of star query results and pick the top-k answers. If all candidate nodes are explored for each candidate combination, the time complexity would be exponential.

We consider the branch and bound pruning technique [19] while traversing among these candidate nodes. To ensure the best quality of candidate selections, we sort each top-k_s result of star queries in Phase 2 in a non-descending order in separate lists. Then we search through each list from the top to do the uniform cost search and construct a search tree. Each path along the root to the leaf node is a matched candidate set for query nodes. While searching from root to leaf, we check the aggregated matching scores and lower and upper bounds along the path. Assume there are top-k candidate node sets with the smallest lower bound score $\underline{F}_{k_{th}}$, by searching the next candidate node and getting it's upper bound lower than $\underline{F}_{k_{th}}$, the node candidate and all the nodes of its subtrees can be pruned.

As shown in the example Fig. 4, we query top-3 nodes v type shown in the part (1) query graph. The data graph has the structure shown in part (2). Initially, all the red nodes are potential answers, when we propagate to node v_4, we obtain the node matching score $S(v_4, u)$. The matching score $S(v_4, u)$ are smaller than the matching scores of top-3 nodes $S(v_1, u)$, $S(v_2, u)$, and $S(v_3, u)$. The node v_4 and its subtrees are all smaller than $S(v_4, u)$, then we just stop propagating and prune them.

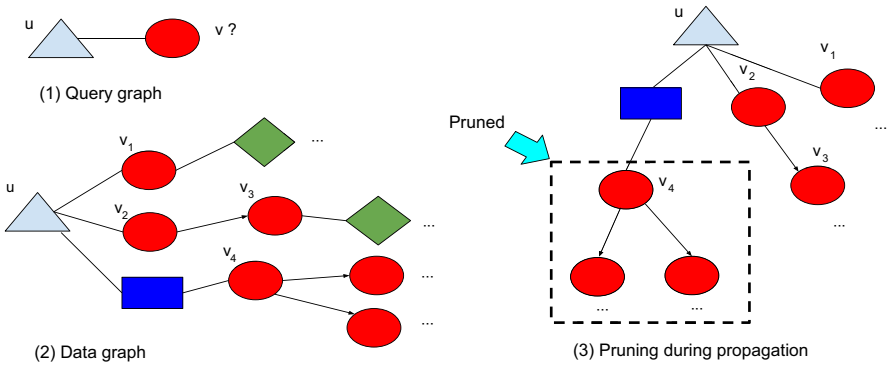


Fig. 4 An example of branch and bound pruning

3.6 Convergence of iteration propagation for graph query

The iteration propagation in essence is a graph traversal problem. The convergence of this graph query is equivalently bounded by the traversal of all the required nodes or no update of the propagation cost. Therefore, two types of iteration conditions are identified to terminate the graph propagation to obtain the final top-k answers.

(1) When all the nodes with designated query node types have been explored or pruned by the bound-based pruning technique, all visited candidate nodes have obtained the necessary matching scores.
(2) When no message is updated for the next propagation, that is, all the candidate nodes' matching scores keep the same as the last iteration.

Lemma 1 *The iterative propagation algorithm of graph query is always converged.*

Proof . The matching score for a graph query is defined as before:

$$F_G(V_Q^U) = \sum_{v \in V_Q^U} S_G(\phi(v)) \tag{10}$$

where

$$S(\phi(v)) = \sum_{v^s \in V_Q^S} r(\phi(v^s), \phi(v)) \tag{11}$$

and

$$r(\phi(u), \phi(v)) = \alpha^{l(\phi(u),\phi(v)) - \beta * |h(\phi(u),\phi(v))|} \tag{12}$$

Therefore, we have:

$$F_G(V_Q^U) \tag{13}$$

$$\begin{aligned} &= \sum_{v \in V_Q^U} S_G(\phi(v)) \\ &= \sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} r(\phi(v^s), \phi(v)) \\ &= \sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} \alpha^{l(\phi(u),\phi(v)) - \beta * |h(\phi(u),\phi(v))|} \end{aligned} \tag{14}$$

Here when traversing along the path in the graph with more and more depths, we assume the shortest distance of $l(\phi(u), \phi(v))$ (abbreviated as l) and $|h(\phi(u), \phi(v))|$ (abbreviated as h) would go bigger and bigger, that is, $l \to \infty$, and $h \to \infty$.

Therefore, to prove that $F_G(V_Q^U)$ converges to a number ϵ, that is, $\lim_{l \to \infty, h \to \infty} F_G(V_Q^U) \to \varepsilon$, we have to show,
$\sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} \alpha^{l(\phi(u),\phi(v)) - \beta * |h(\phi(u),\phi(v))|} \to \varepsilon$
Thus,

$$\begin{aligned} &\lim_{l \to \infty, h \to \infty} \sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} \alpha^{l(\phi(u),\phi(v)) - \beta * |h(\phi(u),\phi(v))|} \\ &= \sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} \lim_{l \to \infty, h \to \infty} \alpha^{l(\phi(u),\phi(v)) - \beta * |h(\phi(u),\phi(v))|} \\ &= \sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} \alpha^{\lim_{l \to \infty, h \to \infty} l(\phi(u),\phi(v)) - \beta * |h(\phi(u),\phi(v))|} \end{aligned} \tag{15}$$

Given a path from node $\phi(u)$ to another node $\phi(v)$, the shortest distance of the path is defined as $l(\phi(u), \phi(v))$, and the hierarchical level distance is defined as $|h(\phi(u), \phi(v))|$. According to the definition of the shortest distance and hierarchical level, we see that $l(\phi(u), \phi(v)) \geq |h(\phi(u), \phi(v))|$, and $\beta \in [0, 1]$, we will have $l(\phi(u), \phi(v)) - \beta * |h(\phi(u), \phi(v))| \geq 0$. Then,

$$\begin{aligned} &\sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} \alpha^{\lim_{l \to \infty, h \to \infty} l(\phi(u),\phi(v)) - \beta * |h(\phi(u),\phi(v))|} \\ &\leq \sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} \alpha^{\lim_{l \to \infty, h \to \infty} 0} \\ &\leq \sum_{v \in V_Q^U} \sum_{v^s \in V_Q^S} 1 \\ &= |V_Q^U| * |V_Q^S| \\ &= \varepsilon \end{aligned} \tag{16}$$

Therefore, we have finally obtained:

$$\lim_{l\to\infty,h\to\infty} F_G(V_Q^U) \to \varepsilon \tag{17}$$

Our iterative propagation for graph query is proved to be converged.

4 Graph query system and implementation

4.1 System architecture

Based on the proposed graph query algorithm with hierarchical relations, we propose the graph query system shown in Fig. 5. It reads the graph data into memory for offline processing. In online processing, given a query graph, it reads the query and decomposes the query graph into one or more star query graphs, then it traverses the data graph according to the defined matching scores and algorithms to obtain candidate nodes for each query node. Finally, the top-k candidate nodes are selected through the candidate selection algorithm as the top-k query result.

4.2 Distributed implementation

To support large information networks, we implement our graph query algorithm in the framework GraphX, which is a distributed graph analytics platform built on Apache Spark [20]. Figure 6 shows our architecture of distributed graph query system. We define a global data structure, Global Vertex State Table (GT) for each vertex stored in the Spark RDD data structure. GT is a user-defined class type which can store the following hash mapping for each anchor node v^a: node type τ, shortest distance sd, hierarchical level difference hd, node closeness score r, closeness score lower bound $\underline{r}$, closeness score upper bound $\overline{r}$, etc. GT values are updated in each iteration of propagation to efficiently decide the bounds for effective pruning of many useless node propagations. We use Google cloud platform [21] to run our

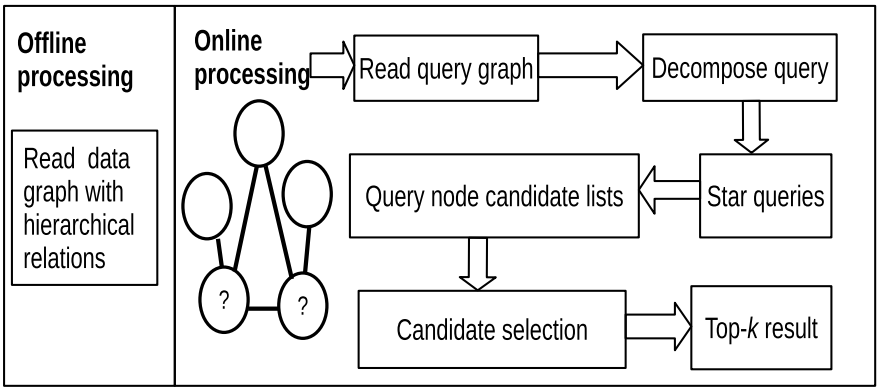


Fig. 5 Graph query system architecture

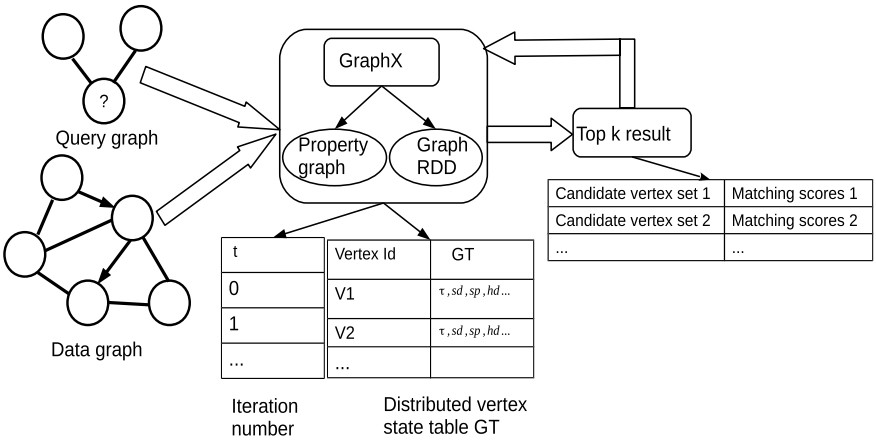


Fig. 6 Distributed graph query implementation architecture

experiment. The basic configuration for spark is composed of 4 worker nodes and each node has 4 CPU cores and 32 G memory. For the experiment of the scalability test, we increase the worker number from 2 to 32.

5 Experimental evaluation

The experiments are designed to answer the questions as follows:

(1) **Effectiveness:** How is the quality of our query algorithm for hierarchical query graphs or mixed query graphs? how is the query with hierarchical inheritance relations compared with state-of-the-art methods?
(2) **Efficiency:** How is the efficiency and scalability of our algorithm on one machine and multiple machines?

We experiment with different parameter values of α and β for matching scores, which show similar results below. In our experimental results, $\alpha = 0.6$ and $\beta = 0.5$ are used.

5.1 Datasets

We experiment on the synthetic data graph, Cisco product data graph, and extended DBLP data graph datasets. Table 1 shows the data statistics for our experiments.

(1) Synthetic data graph: we randomly generate a data graph and create 7 types of nodes. There are 2 attaching node types, 2 inherited node types, and 3 other node types. **(2)** Cisco data graph: we extract the data from its official and related support websites about devices, device properties, etc. The constructed graph schema is shown in Fig. 1a. "Vulnerability" and "Technology" are the attaching node types.

Table 1 Dataset statistics

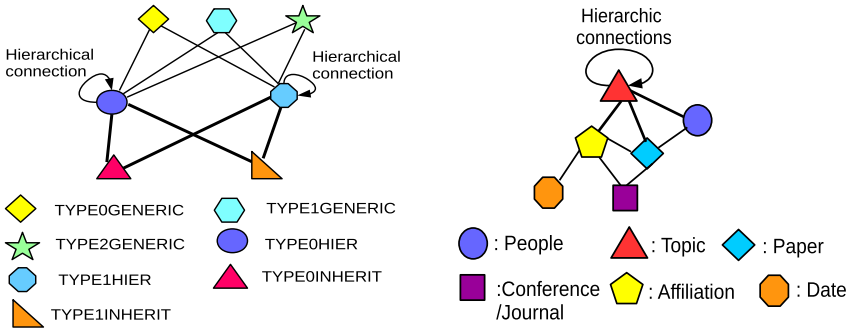
Dataset	|V|	|E|	Avg. degree	No. of vertex types (attaching + inherited + other)
Synthetic Graph Data (Synthetic)	30 M	125.4 M	45	2+2+3
Cisco Product Data (Cisco)	215,347	966,992	12	2+1+4
Extended DBLP Data (DBLP)	1.88 M	25.18 M	58	1+2+9

"Product" is the inherited node type. **(3)** Extended DBLP data graph: it is a DBLP database [22] extending the topics extracted from lists of computer science conferences and journal websites. "Topic" is the attaching node type that is inherited among the conference/journal, paper, and people node types. The data graph schema of the synthetic data graph and extended DBLP data graph are shown in Fig. 7.

5.2 Quality of our graph query

As mentioned earlier, a query graph can be classified into a hierarchical, mixed hierarchical, or non-hierarchical query graph by inheritance relations, and a star query or general query graph based on query node numbers.

We show the results of hierarchical star query graphs and mixed general query graphs here. In each real dataset, one star query example and non-star query example results are shown in Fig. 8. Figure 8a shows the hierarchical star query with all specific nodes as attaching nodes and the query node as an inherited node, and the top-5 query results are found in Cisco data. As seen in the results, different inherited versions of Cisco WebEx meeting server products are queried with higher matching scores. Figure 8b displays different authors with publication papers in a journal and working on the same topic, which is verified to be reasonable online. As the more complex non-star queries shown in Fig. 8c and 8d with each top-1 result, our algorithm GQH can also provide the most relevant query answers.



(a) Synthetic data graph schema (b) Extended DBLP data graph schema

Fig. 7 Data graph schemas of synthetic data graph and extended DBLP data graph

5.3 Comparisons of query quality

Existing state-of-the-art algorithms for graph queries generally only consider the shortest paths and neighbor propagation. Our paper proposes the query with hierarchical inheritance relations improving the recent query algorithm from Jin et al. [9] (GStar query). NeMa [7] is a classical graph query method based on neighborhoods with approximate query. Therefore, we compare the effectiveness of our query algorithm with the graph query based on the GStar query and NeMa query algorithms.

5.3.1 Overall comparison of query quality

We compare GQH with GStar and NeMa Query algorithm using comprehensive query examples on Cisco data and Extended DBLP data. Similar to other existing research work, there is no ground truth available for query quality. We design a metric called inheritance coverage rate (ICR) to measure the overall quality with hierarchical inheritance relations. ICR is the coverage percentage of matched nodes with inheritance relations over the total matched nodes in [0, 1]. For example, given a query graph, a user wants to find a top-5 result of matched nodes. If there are 3 nodes in the hierarchical relations, then ICR = 0.6. The higher the value, the more matched nodes with inherited relations would be found. We create 100 random query graphs to obtain the top-2, top-5, and top-10 results and then calculate the ICR results. We show the average ICR results on the three datasets. It shows the GQH algorithm has better ICR results all the cases on the three datasets than the *GStar* and *NeMA*, in which the query quality has been improved to different extents comprehensively shown in Table 2.

5.3.2 An example of query quality comparison with GStar

We compare GStar with our algorithm GQH based on the example of the query in Fig. 8a and show the result. Table 3 shows top-5 results of comparison with GStar's Query algorithm. GQH shows the possible "Cisco WebEx meeting server version" inheritances as more potential candidates than GStar's query algorithm, with three different numbers of matched candidates. This is because GStar's query algorithm only considers the node types and shortest distances as metrics for ranking.

Table 2 The overall query quality comparisons on the three datasets

	Synthetic			Cisco			Dblp		
	GStar	NeMa	GQH	GStar	NeMa	GQH	GStar	NeMa	GQH
Top-2	0.323	0.389	0.467	0.285	0.279	0.351	0.312	0.305	0.395
Top-5	0.415	0.432	0.533	0.351	0.326	0.416	0.335	0.323	0.458
Top-10	0.435	0.439	0.545	0.383	0.335	0.419	0.348	0.315	0.462

Table 3 One query result of GQH and GStar in the Cisco dataset

	Query result in GQH		Query result in GStar	
Rank	Node	Score	Node	Score
1	Cisco WebEx meetings server versions 0.2	1.7906	Cisco WebEx meetings server versions 1.x	1.7186
2	Cisco WebEx meetings server versions 1.x	1.7100	Cisco WebEx meetings server versions 2.x	1.7186
3	Cisco WebEx meetings server versions 2.x	1.7100	Easy vpn	1.7186
4	Cisco 12000 series spa interface processors running Cisco ios software	1.7015	Cisco unified ip phone	1.7015
5	Cisco xr 12000 series engine 3 line cards	1.7015	Catalyst 6000 supervisor module	1.7015

5.3.3 An example of query quality comparison with NeMa

NeMa in [7] uses nodes' labels and neighborhood similarity in small hops to find the top matched subgraphs. We compare the query quality with our algorithm GHQ for the query in Fig. 8a and show the result in Table 4. It shows top-5 results of comparison with GStar query algorithm. NeMa uses matching cost which measures the cost of matched subgraphs with the query graph. The smaller the cost, the better the matching. "—" indicates no matching result is found, and only top-3 results are returned. It also does not return the hierarchical "Cisco WebEx meeting server" answers. This is because it limits the maximum hops of its visits and does not consider the hierarchical inheritance, which leads to a smaller structural difference but fewer potential matches.

5.4 Efficiency of graph query

We test the efficiency of our GQH algorithm and compare with NeMa and GStar here.

5.4.1 Efficiency of our graph query

We evaluate the efficiency of our graph query system GQH with different top-k values, query graph sizes, and data graph sizes. For each different test, we keep one testing parameter varied and the other unchanged. Each experiment is repeated 20 times and we obtain the average runtime with different parameters.

Varying k: To check how our algorithm scales with different querying k, we examine the average runtime for different top-k values from 1, 2, 5 to 30 in Fig. 9a–c. Three different query sizes [2, 1], [4, 2], and [6, 3] are fixed. It shows the runtime is basically sublinear no matter the k value. This is because the complexity degrees of graphs lead to more than designated top-k answered before the

Table 4 One query result of GQH and NeMa in the Cisco dataset

	Query result in GQH		Query result in NeMa	
Rank	Node	Score	Node	Cost
1	Cisco WebEx meetings server versions 0.2	1.7906	Cisco WebEx meetings server version 0.2	2.0
2	Cisco WebEx meetings server versions 1.x	1.7100	Cisco ASA Series show running-config prior to 7.2.1	2.95833
3	Cisco WebEx meetings server versions 2.x	1.7100	Cisco ASA Series show running-config between 7.2.1 and 8.4	2.95833
4	Cisco 12000 series spa interface processors running Cisco ios software	1.7015	–	–
5	Cisco xr 12000 series engine 3 line cards	1.7015	–	–

termination of iterations. We only fetch the top-k candidates from all the obtained candidates.

Varying query graph size To check how our algorithm scales with different query graph sizes, we examine the average runtime for different query sizes. The query size is defined as a tuple (specific node number, query node number). We select (2, 1), (4, 2) to (10, 10) shown in Fig. 9d–f with top-k value 2, 5, and 10 used. It shows that the running time is basically sublinear with the increase of the query size.

Varying data graph size: We test the query time with varying data graph sizes. We randomly and accumulatively extract subgraphs from the original data graph for different node numbers, covering 10%, 20%, 50%, 80%, and 100%. We measure 3 different query sizes in this scene to obtain the top-20 query time for different graph data sizes. As shown in Fig. 9g–i, the query time also increases sublinearly with the increase of the graph data size.

Scalability on multiple machines: To test the scalability of our GQH algorithm on multiple machines, we test it on Google Cloud Platform (each instance is 4 CPU cores with 32GB memory) to see the average runtime trends with different worker machines deployed. We use 3 different query sizes, with one master and increasing worker machines from 2, 4, to 32 for the top-10 query. Figure 9(j) shows that the running average time decreases sublinearly with the increasing number of workers.

5.4.2 Comparison of efficiency

Here we show some comparisons of efficiency with NeMa and GStar algorithms with different top-k values. We randomly run 100 queries with query graph size = (4, 2) and show the average runtime on the three datasets, which are shown in Fig. 10a–c. Moreover, different query sizes (2, 1), (4, 2) to (10, 10) are compared on the three datasets shown in Fig. 10d–e. It shows that our algorithm has a competitive performance with the other two algorithms. This is because our GQH uses graph traversal and propagation similar to state-of-the-art algorithms. However, our graph query improves the graph query quality with competitive performance.

6 Related work

There exist several classification categories for graph queries. Based on user inputs, it can be classified as keyword query and structured query [7, 23–26]. Based on query answers, it includes exact match and inexact match [15, 27, 28]. Based on matching techniques, it mainly contains indexing-based queries and graph-traversal-based queries for distance, neighbor and random walk [5, 29–33]. Our algorithm focuses on the top-k inexact match for structured graph queries with hierarchical inheritance relations.

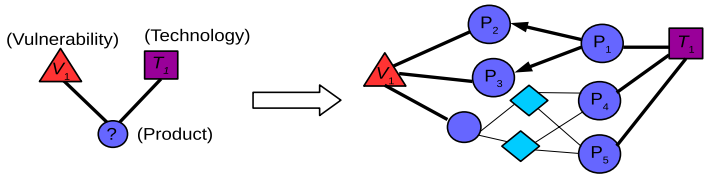
Exact graph matching: The early graph matching/query focuses more on the exact match using subgraph isomorphism matching for exact matching. The subgraph isomorphism matching identifies all the exactly same occurrences in the big graph as the input query subgraph. Most of subgraph isomorphism matching

Fig. 8 Graph queries and their results **a** Hierarchical star query graph in Cisco—Query a product with Vulnerability V_1 and used Technology T_1, and its top-5 results in order. **b** Mixed star query graph in DBLP – Query an author that works on Topic T_1 and cooperates with Person Pp_1, and its top-5 results in order. **c** Mixed general query graph in Cisco—Query a product that has Vulnerability V_1 and V_2, another product that belongs to a workgroup W_1, and their common technology also used in a product P_1, and the top-1 result. (d) Mixed general query graph in DBLP – Query an author that works on Topic T_1 and T_2, cooperates with another person that works on Topic T_3 and published a paper coauthored with Person Pp_1 in the year D_1, and its top-1 result

methods use sub-structures-based indexing techniques for efficient searching [34–36, 36–39]. Some of the impactful methods are Ullmann's backtracking method [34], VF and VF2 method [36, 40] and SwiftIndex [41]. Ullmann's backtracking method is one of the earliest algorithms to explore subgraph isomorphism and utilizes the backtracking to inferentially eliminate successor nodes in the tree search. Without using the topology of the large graph and making the query in the large graph applicable, VF and VF2 method [36, 40] reduce the search space and store the information of the state space search in more effective data structures to significantly reduce the matching time and the memory requirements. The SwiftIndex method [41] tries to reduce the search space for large query graphs with bounding and frequencies of features and a new feature-based index technique for the filtering stage.

Inexact graph matching: In addition to considering graph query with exact matches, various techniques have been proposed for structured graph queries to obtain inexact matches for large graph applications [7, 7, 8, 10, 10, 27, 28, 42–44]. Indexing-based propagation techniques are mostly utilized to find the inexact match in a graph for a query graph. In the era of big data, recent work allows users to express their own input query as a structured query graph and to do the graph traversal based on node and path similarity for matching. Ness [27] finds the inexact graph match with the neighborhood similarity search. It converts a large graph network into a set of multidimensional vectors based on available sophisticated indexing and similarity search algorithms. NeMa [7] and SLQ [10] consider different similarity transformations for node to match the query graph and subgraphs in a data graph. Su et al. [44] consider the graph query based on user relevance to further improve the query quality. Some other research [25, 45] considers the multiple attributes of nodes for graph query. Jin et al. [8, 9] propose a specified ranking function for a structured graph query with specific nodes to find answer nodes. Most of them use indexing which takes large spaces, or graph traversal with only two dimensions of node and edge similarities. We consider one more dimension of hierarchical inheritance relations for effective queries.

Top-*k* graph query: Top-*k* graph query tries to obtain top-*k* matched answers for the graph query. The common practice for top-*k* search is to use threshold algorithms to find the top matches by traversing a sorted node/edge list [16, 46–51]. They mostly require precomputed and sorted lists to derive the bounds with indexing and relevance functions to get top interesting results [47, 52, 53]. Gupta et al. [53] optimize the time-consuming ranking-after-match for obtaining the top-*k* query. They introduce topology index and graph maximum meta-path weight index structures for the network, which are both computed offline. Then,

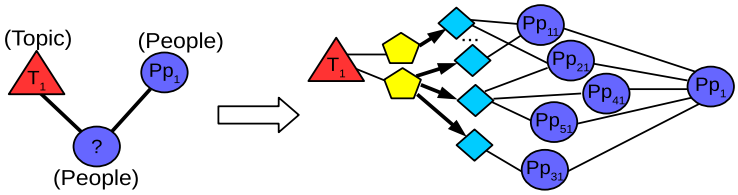


V_1 : Cisco asa software ikev1 and ikev2 buffer overflow vulnerability

T_1 : Voice - communications manager additional apps and plugins

P_1: Cisco webex meetings server versions 0.2
P_2: Cisco webex meetings server versions 2.x
P_3: Cisco webex meetings server versions 1.x+++0
P_4: Cisco 12000 series gigabit switch router (gsr)
P_5: Cisco xr 12000 series engine 3 line cards

(a)

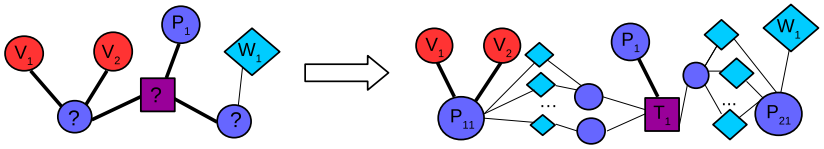


T_1: Data semantics and integrity Constraints

Pp_1: Youngjoon Hong

Pp_{11} : Roger Temam
Pp_{21} : Joris Van Deun
Pp_{31} : Chang-Yeol Jung
Pp_{41} : Gung-Min Gie
Pp_{51} : Arthur Bousquet

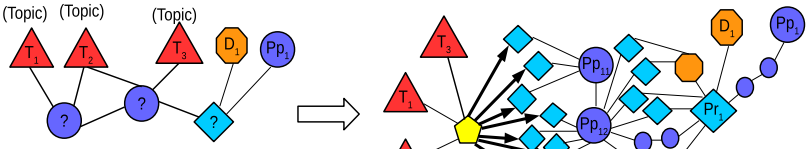
(b)



V_1: Multiple vulnerabilities in ntpd
V_2: Vulnerability in Java deserialization
P_1: Cisco telepresence system 3200
W_1: licensing-man-us

P_{11}: Cisco network analysis module
T_1: Telepresence
P21: Cisco catalyst 6500 series firewall services module

(c)



T_1: Foundations and concept formalization

T_2: Domain specific conceptual modeling languages, methods and frameworks

T_3: Methodologies and tools for conceptual design

D_1: 2006 Pp_1: Weizhu Bao

Pp_{11} : Andrew J. Wathen Pp_{12} : Claude Brezinski

Pr_1 : A small note on the scaling of symmetric positive definite semiseparable matrices

(d)

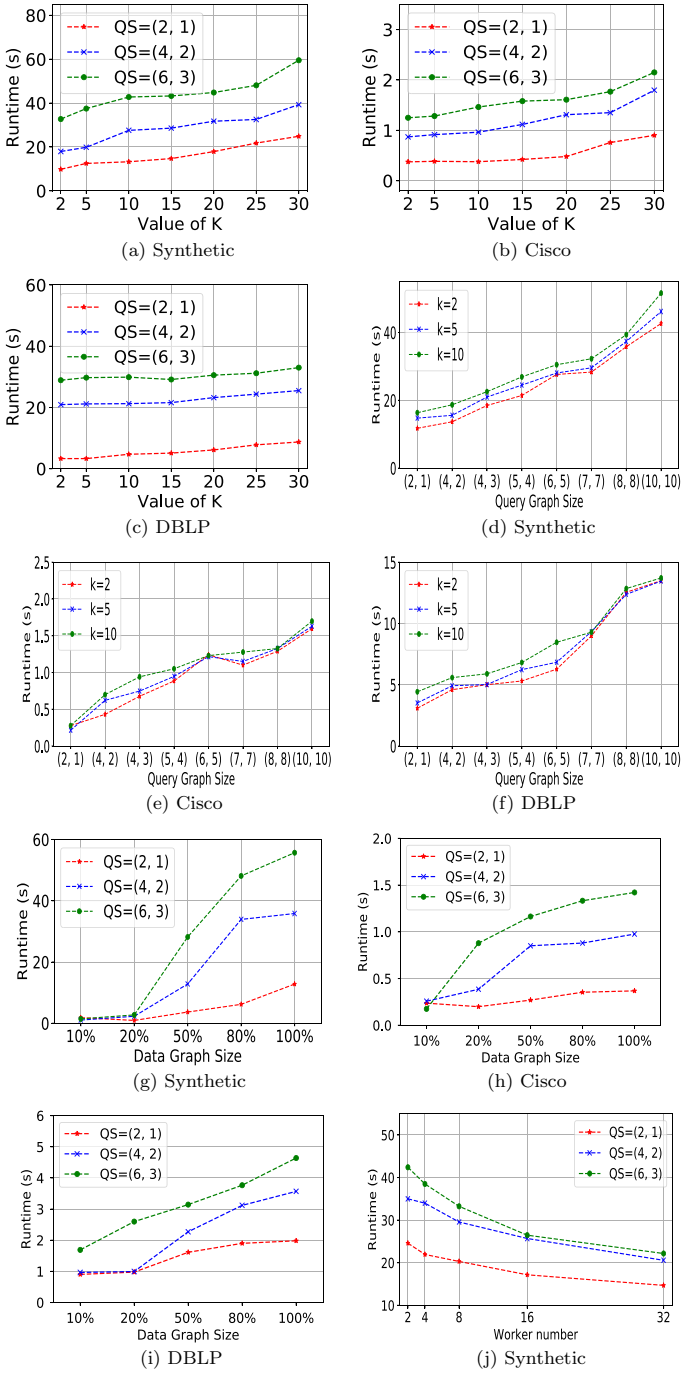
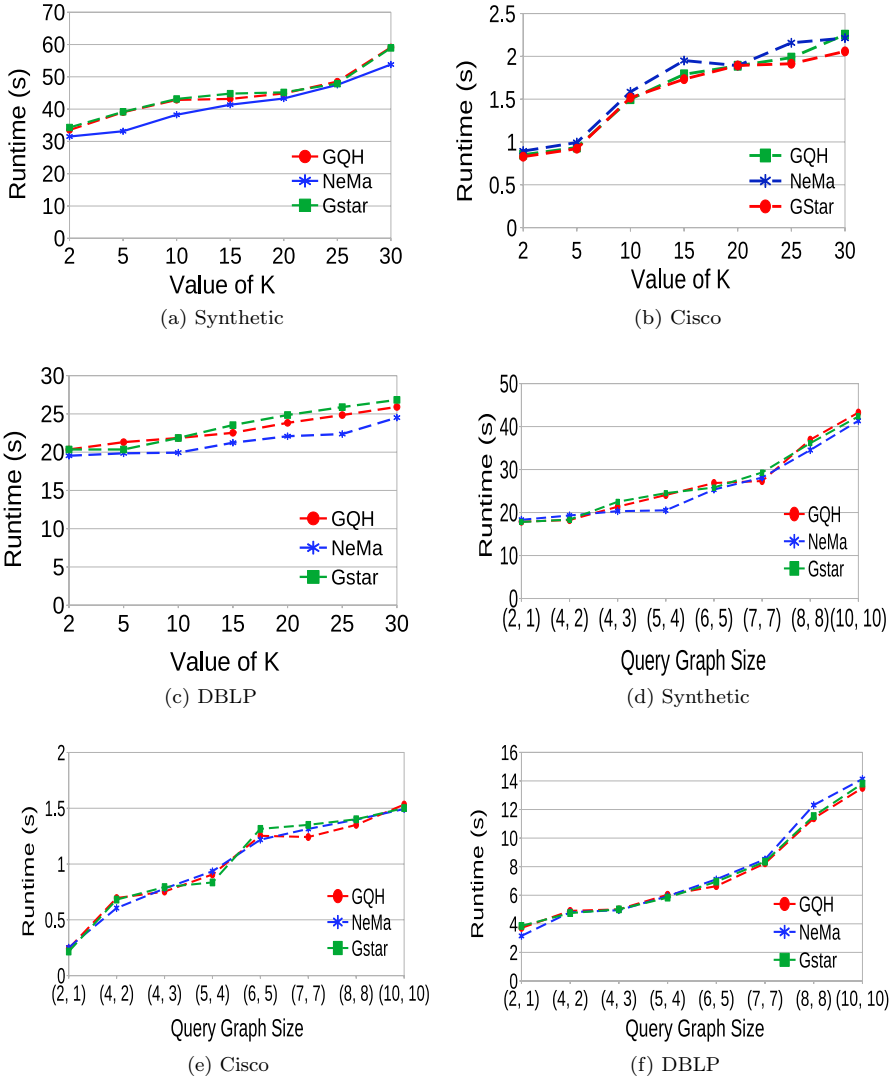


Fig. 9 Efficiency and scalability of GQH on three datasets



(a) Synthetic (b) Cisco

(c) DBLP (d) Synthetic

(e) Cisco (f) DBLP

Fig. 10 Efficiency comparisons on three datasets with different top-k values and query sizes

these indexes are utilized for answering interesting subgraph queries online efficiently for the top-k answers. Yang et al. [16] consider the STAR-query structure and top-k ranked join for general graph queries, but the matches are limited to answer subgraphs with paths of bounded length. Jin et al. [8, 9] consider the shortest path propagation with a specified matching score function for structured graph query to find top-k answer nodes. Our algorithm extends the work and considers the top-k general graph with an efficient ranking score and bounded-based solution without the limitation of path lengths for hierarchical relation inheritance.

7 Conclusion

We consider an additional dimension of hierarchical inheritance relations on real-world heterogeneous information networks for graph query. The problem is reformulated with hierarchical inheritance relations, and we propose a graph query algorithm based on that for star query and general graph query. With the bounding-based techniques, our algorithm can effectively capture hierarchical inheritance relations on information networks for better query answers, and competitive performances are also achieved.

Acknowledgements This work was supported in part by National Science Foundation Grants CNS-1815412 and CNS-1908536.

References

1. Davis, D., Lichtenwalter, R., Chawla, N.V.: Multi-relational link prediction in heterogeneous information networks. In: 2011 International Conference on Advances in Social Networks Analysis and Mining, pp. 281–288. IEEE (2011)
2. Gupta, M., Gao, J., Yan, X., Cam, H., Han, J.: On detecting association-based clique outliers in heterogeneous information networks. In: 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), pp. 108–115. IEEE (2013)
3. Shi, C., Li, Y., Zhang, J., Sun, Y., Philip, S.Y.: A survey of heterogeneous information network analysis. IEEE Trans. Knowl. Data Eng. **29**(1), 17–37 (2017)
4. Sun, Y., Han, J.: Mining heterogeneous information networks: principles and methodologies. Synth. Lect. Data Mining Knowl. Discov. **3**(2), 1–159 (2012)
5. Fujiwara, Y., Nakatsuji, M., Onizuka, M., Kitsuregawa, M.: Fast and exact top-k search for random walk with restart. Proc. VLDB Endowment **5**(5), 442–453 (2012)
6. Fujiwara, Y., Nakatsuji, M., Shiokawa, H., Mishima, T., Onizuka, M.: Fast and exact top-k algorithm for pagerank. In: Twenty-Seventh AAAI Conference on Artificial Intelligence (2013)
7. Khan, A., Wu, Y., Aggarwal, C.C., Yan, X.: Nema: Fast graph search with label similarity. In: Proceedings of the VLDB Endowment, vol. 6, pp. 181–192. VLDB Endowment (2013)
8. Jin, J., Khemmarat, S., Gao, L., Luo, J.: Querying web-scale information networks through bounding matching scores. In: Proceedings of the 24th International Conference on World Wide Web, pp. 527–537. International World Wide Web Conferences Steering Committee (2015)
9. Jin, J., Luo, J., Khemmarat, S., Dong, F., Gao, L.: Gstar: an efficient framework for answering top-k star queries on billion-node knowledge graphs. World Wide Web **22**(4), 1611–1638 (2019)
10. Yang, S., Wu, Y., Sun, H., Yan, X.: Schemaless and structureless graph querying. Proc. VLDB Endowment **7**(7), 565–576 (2014)
11. Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., Zhang, H.: Fast approximate nearest-neighbor search with k-nearest neighbor graph. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)
12. Long, D.P., Garigliano, R.: Inheritance in hierarchical relational structures. In: Proceedings of the 12th conference on Computational linguistics-Volume 1, pp. 384–386. Association for Computational Linguistics (1988)
13. Clauset, A., Moore, C., Newman, M.E.: Hierarchical structure and the prediction of missing links in networks. arXiv preprint arXiv:0811.0484 (2008)
14. Jiang, J.Y., Cheng, P.J., Lin, C.Y.: Entity-driven type hierarchy construction for freebase. In: Proceedings of the 24th International Conference on World Wide Web, pp. 47–48. ACM (2015)
15. Lee, J., Han, W.S., Kasperovics, R., Lee, J.H.: An in-depth comparison of subgraph isomorphism algorithms in graph databases. In: Proceedings of the VLDB Endowment, vol. 6, pp. 133–144. VLDB Endowment (2012)

16. Yang, S., Han, F., Wu, Y., Yan, X.: Fast top-k search in knowledge graphs. In: Data Engineering (ICDE), 2016 IEEE 32nd International Conference on, pp. 990–1001. IEEE (2016)
17. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. VLDB J. Int. J. Very Large Data Bases **13**(3), 207–221 (2004)
18. Khemmarat, S., Gao, L.: Fast top-k path-based relevance query on massive graphs. IEEE Tran. Knowl. Data Eng. **28**(5), 1189–1202 (2016)
19. Clausen, J.: Branch and bound algorithms-principles and examples, pp. 1–30. Department of Computer Science, University of Copenhagen pp (1999)
20. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: Graphx: A resilient distributed graph system on spark. In: First International Workshop on Graph Data Management Experiences and Systems, p. 2. ACM (2013)
21. Bisong, E.: An overview of google cloud platform services. Building Machine Learning and Deep Learning Models on Google Cloud Platform pp. 7–10 (2019)
22. Ley, M.: Dblp computer science bibliography. http://dblp. uni-trier. de/ (2008)
23. Kargar, M., Golab, L., Szlichta, J.: egraphsearch: Effective keyword search in graphs. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pp. 2461–2464 (2016)
24. Han, S., Zou, L., Yu, J.X., Zhao, D.: Keyword search on rdf graphs-a query graph assembly approach. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 227–236 (2017)
25. Roy, S.B., Eliassi-Rad, T., Papadimitriou, S.: Fast best-effort search on graphs with multiple attributes. IEEE Trans. Knowl. Data Eng. **27**(3), 755–768 (2015)
26. Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. Proc. VLDB Endowment **12**(5), 461–474 (2019)
27. Khan, A., Li, N., Yan, X., Guan, Z., Chakraborty, S., Tao, S.: Neighborhood based fast graph search in large networks. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 901–912. ACM (2011)
28. Mongiovi, M., Di Natale, R., Giugno, R., Pulvirenti, A., Ferro, A., Sharan, R.: Sigma: a set-cover-based inexact graph matching algorithm. J. Bioinformatics Comput. Biol. **8**(02), 199–218 (2010)
29. Zheng, W., Zou, L., Feng, Y., Chen, L., Zhao, D.: Efficient simrank-based similarity join over large graphs. Proc. VLDB Endowment **6**(7), 493–504 (2013)
30. Dave, V.S., Al Hasan, M.: Topcom: Index for shortest distance query in directed graph. Database and expert systems applications., pp. 471–480. Springer, Berlin (2015)
31. Ren, X., Wang, J.: Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. Proc. VLDB Endowment **8**(5), 617–628 (2015)
32. Wu, Y., Jin, R., Zhang, X.: Efficient and exact local search for random walk based top-k proximity query in large graphs. IEEE Trans. Knowl. data Eng. **28**(5), 1160–1174 (2016)
33. Zhao, P., Han, J.: On graph query optimization in large networks. Proc. VLDB Endowment **3**(1–2), 340–351 (2010)
34. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM (JACM) **23**(1), 31–42 (1976)
35. Giugno, R., Shasha, D.: Graphgrep: A fast and universal method for querying graphs. In: 2002 International Conference on Pattern Recognition, vol. 2, pp. 112–115. IEEE (2002)
36. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub) graph isomorphism algorithm for matching large graphs. IEEE Trans. Pattern Anal. Mach. Intell. **26**(10), 1367–1372 (2004)
37. Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: 2002 IEEE International Conference on Data Mining, 2002. Proceedings., pp. 721–724. IEEE (2002)
38. Yan, X., Yu, P.S., Han, J.: Graph indexing: a frequent structure-based approach. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 335–346 (2004)
39. Yan, X., Yu, P.S., Han, J.: Substructure similarity search in graph databases. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 766–777 (2005)
40. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In: 3rd IAPR-TC15 workshop on graph-based representations in pattern recognition, pp. 149–159. Citeseer (2001)
41. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. Proc. VLDB Endowment **1**(1), 364–375 (2008)
42. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognit. Lett. **1**(4), 245–253 (1983)

43. Cesar, R.M., Jr., Bengoetxea, E., Bloch, I., Larrañaga, P.: Inexact graph matching for model-based recognition: evaluation and comparison of optimization algorithms. Pattern Recognit. **38**(11), 2099–2113 (2005)
44. Su, Y., Yang, S., Sun, H., Srivatsa, M., Kase, S., Vanni, M., Yan, X.: Exploiting relevance feedback in knowledge graph search. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144. ACM (2015)
45. Du, B., Zhang, S., Cao, N., Tong, H.: First: Fast interactive attributed subgraph matching. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1447–1456. ACM (2017)
46. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4), 614–656 (2003)
47. Fan, W., Wang, X., Wu, Y.: Diversified top-k graph pattern matching. Proc. VLDB Endowment **6**(13), 1510–1521 (2013)
48. Zou, L., Chen, L., Lu, Y.: Top-k subgraph matching query in a large graph. In: Proceedings of the ACM first Ph. D. workshop in CIKM, pp. 139–146 (2007)
49. Wei, Z., He, X., Xiao, X., Wang, S., Shang, S., Wen, J.R.: Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In: Proceedings of the 2018 International Conference on Management of Data, pp. 441–456 (2018)
50. Semertzidis, K., Pitoura, E.: Top-*k* durable graph pattern queries on temporal graphs. IEEE Trans. Knowl. Data Eng. **31**(1), 181–194 (2018)
51. Zhu, Y., Qin, L., Yu, J.X., Cheng, H.: Answering top-k graph similarity queries in graph databases. IEEE Trans. Knowl. Data Eng. **32**(8), 1459–1474 (2019)
52. Cheng, J., Zeng, X., Yu, J.X.: Top-k graph pattern matching over large graphs. In: Data Engineering (ICDE), 2013 IEEE 29th International Conference on, pp. 1033–1044. IEEE (2013)
53. Gupta, M., Gao, J., Yan, X., Cam, H., Han, J.: Top-k interesting subgraph discovery in information networks. In: Data Engineering (ICDE), 2014 IEEE 30th International Conference on, pp. 820–831. IEEE (2014)