2017

Query on Knowledge Graphs with Hierarchical Relationships

Kaihua Liu
University of Massachusetts Amherst

QUERY ON KNOWLEDGE GRAPHS WITH HIERARCHICAL RELATIONSHIPS

A Thesis Presented

by

KAIHUA LIU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

September 2017

Department of Electrical and Computer Engineering

QUERY ON KNOWLEDGE GRAPHS WITH HIERARCHICAL RELATIONSHIPS

A Thesis Presented

by

KAIHUA LIU

Approved as to style and content by:

____

Lixin Gao, Chair

____

Weibo Gong, Member

____

David Irwin, Member

____

C.V.Hollot, Department Head
Department of Electrical and Computer Engineering

ACKNOWLEDGEMENTS

I own the deepest gratitude to Professor Lixin Gao. She offered me with this intriguing research project and every possible resource from the beginning. I would also like to extend my appreciation to Professor Weibo Gong and Professor David Irwin. Thank you for being the member of my thesis committee. Thank you for your valuable time, comments and encouragement.

I appreciate my colleague Fubao Wu for their generous help in the completion of this thesis. Fubo helped me with improving writings and experiments of my thesis. We also discussed my algorithms and results analysis. These technical supports were crucial parts in this thesis.

In the end, I will never forget the care from my parents who I miss every day. Forever you are my strongest backup and the essential power for my constant endeavor.

ABSTRACT

QUERY ON KNOWLEDGE GRAPHS WITH HIERARCHICAL RELATIONSHIPS

SEPTEMBER 2017

KAIHUA LIU

B.S., ANHUI JIANZHU UNIVERSITY

M.S.E.C.E, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Lixin Gao

The dramatic popularity of graph database has resulted in a growing interest in graph queries. Two major topics are included in graph queries. One is based on structural relationship to find meaningful results, such as subgraph pattern match and shortest-path query. The other one focuses on semantic-based query to find question answering from knowledge bases. However, most of these queries take knowledge graphs as flat forms and use only normal relationship to mine these graphs, which may lead to mistakes in the query results. In this thesis, we find hierarchical relationship in the knowledge on their semantic relations and make use of hierarchical relationship to query on knowledge graphs; and then we propose a meaningful query and its corresponding efficient query algorithm to get *top-k* answers on hierarchical knowledge graphs. We also design algorithms on distributed frameworks, which can improve its performance. To demonstrate the effectiveness and the efficiency of our algorithms, we use CISCO related products information that we crawled from official websites to do experiments on distributed frameworks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

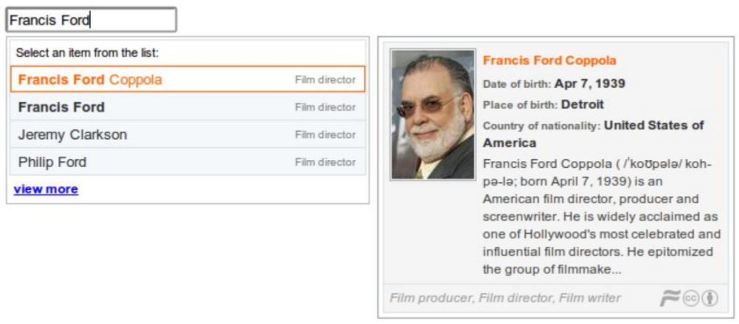
CHAPTER 1

INTRODUCTION TO KNOWLEDGE GRAPH QUERIES AND CONTRIBUTION

This chapter presents an overview of up-to-date graph query research, which is related to our graph query idea and algorithms. And we also propose our ideas and contribution to query on knowledge graphs with hierarchical relationships.

1.1 Knowledge graphs

Large scale networks, including google knowledge graphs provide our raw materials to answer real-life questions. Besides google knowledge graphs, many popular knowledge graphs like DBPedia [1] and Freebase [2] which collect useful knowledge and facts of the work into information networks. For example, freebase is a very popular knowledge base used for research. Freebase is a pratical and scalable tuple database used to maintain general human knowledge. It contains more than 125,000,000 tuples, more than 4000 types and more than 7000 properties. These tuples provide the chance to build data-oriented application including graph search engine. The key components of freebase includes: A scalable Tuple Store, An HTTP/JSON-Based API, lightweight and collaborative type system, large and various data set and standard rules in formulation. Figure 1 is an example knowledge about *Francis Ford* showed on web service.


Francis Ford
Select an item from the list:
Francis Ford Coppola Film director
Francis Ford Film director
Jeremy Clarkson Film director
Philip Ford Film director
view more

Francis Ford Coppola

Date of birth: Apr 7, 1939

Place of birth: Detroit

Country of nationality: United States of America

Francis Ford Coppola (/ˈkoʊpələ/ koh-pə-lə; born April 7, 1939) is an American film director, producer and screenwriter. He is widely acclaimed as one of Hollywood's most celebrated and influential film directors. He epitomized the group of filmmake...

Film producer, Film director, Film writer

Figure 1: Freebase example

1.2 Overview of research on graph query

Knowledge graph(KG) queries become a very popular topic in research, lots of types of searches have been developed, such as shortest-path search [9, 3], reachability search [8], and pattern match query [24, 5, 12]. Some knowledge graph-based queries also incorporate semantic parsing to support open-domain question answering(QA) [22] and provide a deep understanding of questions. Graph searches show meaningful results based on relationships among knowledge. However, most of queries mentioned above just take KGs as a flat graph, and use connections among vertices of the KG to find answers. For instance, among these queries, pattern match query is often discussed. In the pattern match query, we have a data graph G with m vertices, a query graph Q with n vertices, and a parameter δ; If m vertices in G have similar labels and adjacent conditions as Q, and corresponding distances are no larger than δ, this subgraph is the one that matches Q. Apparently, only node types and connectivity conditions are considered in these queries. But when we consider semantic meanings or other relationships into knowledge relations, hierarchy appears in part of knowledge graphs, which means some nodes are a subset of others. Hierarchical information is very common in all the information around us like relationship among animal spices, so we can make good use of this kind of relations to find more meaningful results.

This chapter covers our ideas to search on knowledge graphs with hierarchical relationships. We also represent related example and analysis to prove our ideas in building and searching on knowledge graphs considering hierarchical relationships.

1.3 Knowledge graph query with hierarchical relationships

1.3.1 Knowledge graph and hierarchical structure formulation

As we talked above, current popular knowledge graphs like freebase are just to connect information items based on their facts. The way that they store related information items is Tuple Store which helps scalability of data creation and maintenance.

The knowledge graphs we use here needs to be formulated into hierarchical structure. In this kind of knowledge graphs, two major relationships exist and we use the type of edge to represent this relationship. The first one is hierarchical relationship, two nodes which have this kind of relationship means that one node is the super class of the other. Another relationship is non-hierarchical relationship, this kind of relationship is the same as connections in the freebase, which just represents correlated information items. We use adjacent lists to store graphs, including edge types, for further usage.

1.3.2 Query on knowledge graphs example

In this thesis, we propose a new query doing search on knowledge graphs with hierarchical relationships, we also provide corresponding efficient ranking algorithm and implementation. Searches on this kind of knowledge graphs provide more meaningful results than most of up-to-date graph searches and traditional search engines, it will use the hierarchical relationship to get more interesting and trustworthy information.

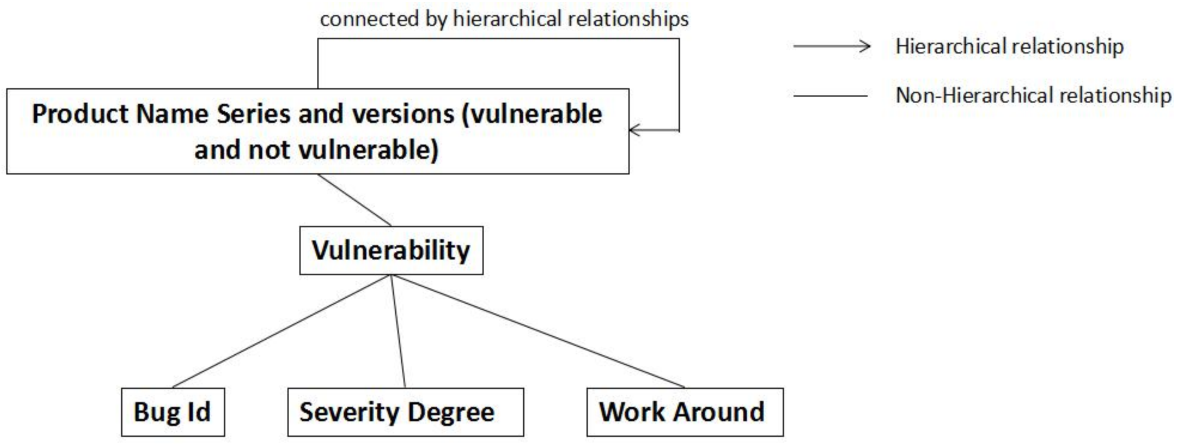
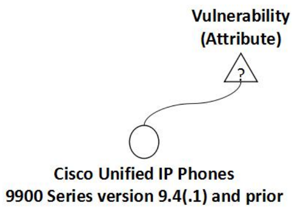

connected by hierarchical relationships
Hierarchical relationship
Non-Hierarchical relationship
Product Name Series and versions (vulnerable and not vulnerable)
Vulnerability
Bug Id
Severity Degree
Work Around

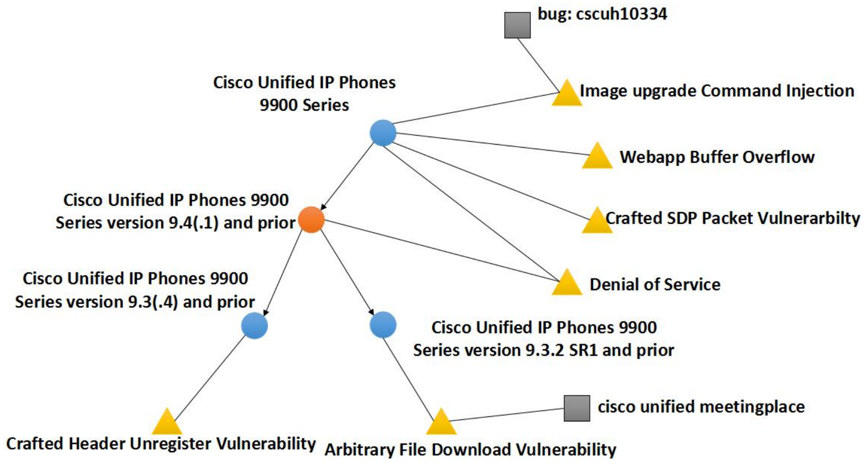
Figure 2 Knowledge graph schema

We provide an example to show the concept of query on knowledge graphs with hierarchical relationships. Figure 2 is the schema of the knowledge graph that we use to do experiments, it includes CISCO products and their properties. The majority of nodes with hierarchical relationship are different CISCO products series, CISCO product series that is located at higher hierarchy covers all other products and these relationships. The hierarchical relationships in this knowledge graphs can be recognized directly by semantic meanings. In the schema, hierarchical relationship connects products at different levels. Products located at higher level are more general, and lower level products are subsets of their connected higher level products. In Figure 3(b), *Cisco Unified IP Phones 9900* is located at higher level and it is subset of *Cisco Unified IP Phones 9900 Series.*

Using CISCO products knowledge graph, we gives an example to show the usage of hierarchical relationships and the essence of our query on graphs. Figure 3(a) is a query graph, given a product *CISCO product Cisco Unified IP Phones 9900 Series Firmware version 9.4(.1) and prior*, and we want to search for all possible vulnerabilities that affect this product. Figure 3(b) is the data graph related to this query. In Figure 3(b), red node is the source node, blue nodes are intermediate nodes during the query, yellow nodes are target nodes and gray nodes are other unrelated nodes during the query. Obviously, several answers are presented. The first category is confirmed answers, which are directly connected to the source node or attributes of source node's ancestors, for instance *Cisco Unified IP Phone 9900 Series Denial of Service Vulnerability* and *Cisco Unified IP Phone 8900/9900 Series Crafted SDP Packet Vulnerability*; The other category is potential answers which are also tightly connected to the source node, but we can't make sure of their correctness; These kind of answers may locate at lower level or siblings of the source node's ancestors, so we predict them as vulnerabilities may not be detected, such as *Cisco 9900 Series Phone Arbitrary File Download Vulnerability*. Potential answers are ranked by their relevant positions in hierarchy and shortest distance on the graph.



(a) Query Graph


bug: cscuh10334
Image upgrade Command Injection
Cisco Unified IP Phones
9900 Series
Webapp Buffer Overflow
Cisco Unified IP Phones 9900
Series version 9.4(.1) and prior
Crafted SDP Packet Vulnerarbilty
Denial of Service
Cisco Unified IP Phones 9900
Series version 9.3(.4) and prior
Cisco Unified IP Phones 9900
Series version 9.3.2 SR1 and prior
cisco unified meetingplace
Crafted Header Unregister Vulnerability
Arbitrary File Download Vulnerability

(b) Part of data graph related to the query graph

Figure 3 Hierarchy-based query and answer example

1.4 Summary of contributions

In our work, our goal is to make good use of knowledge graphs with hierarchical relationships and design efficient algorithms at this situation. To summarize, the key contributions are as follows,

(1) We propose a new way to construct knowledge graphs. In our knowledge graphs, information entities are not only simply connected, but they are also formulated if hierarchical relationships exist.

(2) We present a new query on knowledge graphs including hierarchical relationship, and an effective ranking score, which contains both distance and relative position in hierarchy factors on knowledge graphs.

(3) We give an efficient implementation strategy of the ranking algorithm on distributed system. Combined with the ranking algorithm, we use bounding ranking scores to find *top-k* results fast.

(4) In the evaluation part, we do experiments on both single source query and star query, and compare our algorithm with baseline ranking algorithm which only makes distance as the key factor. Then we use the ground truth to prove our effectiveness of our query algorithms.

CHAPTER 2

PROBLEM DEFINITION

In knowledge graphs, each node represents an information entity; Nodes in the graph have multiple types; Edges have types that depends on types of nodes which they connect and contain relationships information between two nodes on each end, hierarchical relationship or non-hierarchical relationship. In this chapter, we gives a detailed definition of graph queries on hierarchical knowledge graphs and the problem we solve in this kind of graph query.

2.1 Hierarchical knowledge graph modeling

The knowledge graph is modeled as a graph $G(V_G, E_G, l_G(V), l_G(E))$, where V_G is a set of vertices with labeling and E_G is a set of edges with labeling in the graph. $l_G(V)$ denotes the vertex labeling function which maps the label to information entities. $l_G(E)$ denotes the edge labeling function. The size of G is defined as $|V_G|$, referring to the size of vertex set. At some situations, two nodes have no hierarchical relationship and we just label it as non-hierarchical relationship, such as the relation between CISCO products and their vulnerability attributes.

As is the schema of the hierarchical knowledge graph in Figure 2, for a product name, it may connect to a more general series product name or more specific products in a hierarchical way, and it may also have relationship to its vulnerabilities and bugs. Each vulnerability also has its attributes, such as bug ids, workarounds and critical degrees. A product series name may also connect to other series which have attributes in common. It represents a kind of weak relevance without hierarchical property. Using these connections, we can find complete information including both direct and indirect correlation to the given information.

2.2 Query graph illustration

A graph G_{sub} is a subgraph of G, which contains a part of connected nodes in graph G. We define a query graph using its all source nodes V_s and type of target nodes t, then describe it as $Q(V_s, t)$. During the process of querying for answers, we can generate G_{sub} derived from source nodes. Answers are located at the margin of subgraphs.

In Figure 3(a), we give a query graph. We want to find confirmed vulnerabilities and potential vulnerabilities of *Cisco Unified IP Phones 9900 Series Firmware version 9.4(.1) and prior* on a hierarchical knowledge graph. Figure 3(b) is the subgraph related to this product. The candidate answers of query graph are located at the margin of the subgraph in Figure 3(b). As for the difference between confirmed vulnerabilities and potential vulnerabilities, confirmed ones should be ancestors of the source node and can be confirmed to be correct based on it hierarchical relationship; But potential ones are only related in space and has no strong hierarchical relationships. Different vulnerabilities are distinguished by ranking score. The method of ranking will be illustrated in the next chapter.

Obviously, vulnerability *Cisco Unified IP Phone 9900 Series Denial of Service Vulnerability* can be found on traditional search engine, like Google. Based on the hierarchical relationship, vulnerabilities like *Cisco Unified IP Phone 8900/9900 Series Crafted SDP Packet Vulnerability*, *Cisco Unified IP Phones 9900 Series Image Upgrade Command Injection Vulnerability* and so on, can also be found as confirmed results, because these attributes have broader effects on *IP Phones products*, including *Unified IP Phone 9900 Series*. The vulnerability like *Cisco 9900 Series Phone Arbitrary File Download Vulnerability* may also be found as potential risk, because of closeness in distance on the hierarchical knowledge graph.

2.3 Answers to the hierarchy-based query

Given a knowledge graph with hierarchical relationship, and a query graph with known data and target data attributes, the problem is to find all possible target nodes with rankings. The answers are defined as all the nodes that meet target data attribute requirements and directly connected to or several hops away from the source node. Edges along the path from a source node to a target node include both types of edges, change of hierarchy and attribute related type. They are used to determine ranking scores on each step. In order to terminate searching subgraphs from source nodes, the termination condition is defined as reaching to the wanted types of nodes or the end of graph.

In this work, we also propose the *top-k* hierarchy-based query problem, which returns the most correlated k answers that are more likely to be ones we want. The ranking scores show both the relevant position to the source node in hierarchical structure and the distance away from the source node. Answers with higher ranking scores should be more preferred ones.

CHAPTER 3

METHODOLOGY IN RANKING CANDIDATES AND ACCELERTING QUERIES

In this chapter, we give a formal presentation of the algorithm in searching and ranking results found on knowledge graphs. To benefit the description of our algorithm, we do research on the undirected knowledge graph with labeled vertices and edges. The labels of edges are used to distinguish different relationships. For convenience, we use "graph" to represent the knowledge graph in this section.

3.1 Ranking score function

3.1.1 Baseline ranking score function

Most of current knowledge graph query algorithms only consider distance and the number of shortest paths into ranking scores. For a target $node\, u$ and a query graph $Q(V_s, t)$, the type of $node\, u$ is t, and its baseline ranking scores specifies how relevant it is to each source node in V_s in query graph, defined as follows:

$$R_{baseline}(u) = \sum_{v \in V_s} \alpha^{\frac{l_{(u,v)}}{N(u,v)}} \qquad u \neq v$$

Where α is a constant between 0 and 1, V_s is the set of source nodes in the query graph $Q(V_s, t)$, $l(u,v)$ is the length of the shortest path between $node\, u$ and $node\, v$, $N(u,v)$ is the number of shortest paths. Obviously $R_{baseline}(u)$ is larger when node u has shorter length to each source node or more shortest paths between two nodes. According to the baseline ranking score function, we can find query results located at closer to each source node with higher rank; If there are more shortest paths between two nodes, the ranking score is higher.

3.1.2 Hierarchy definition

In a graph, we can find several candidates during the search, so ranking them becomes necessary. The ranks of candidates should base on closeness to the known data in both the structure of graph and semantic realities. Figure 4 is an example of a subgraph, we use this example to illustrate our ranking rules. According to facts in graphs, answers which are directly connected to the source node should be ranked at first like Node A in Figure 4, it can also be found directly on Google. Then results which are superclass of the source node, their properties are more likely to be the results ranked the same as directly-connected candidates. Candidates which are superclass of known data, should also be taken as reliable candidates, like Node B and Node C in Figure 4. Finally, potential candidates should be ranked, and two situations should be considered. One is like Node E in Figure 4, it is the sibling of the source node's ancestors. The other one is like Node D in Figure 4, it is the subclass of the source node. The reason why they are potential candidates is that they are just connected indirectly, and have no reliable relationship based on hierarchy.

Here, we describe how to give each result a ranking score. For $node\,u$ found as an answer and related to $node\,v$ in the graph, its hierarchy relative to the known data is defined as its own hierarchy or the hierarchy of its parent node. We define the hierarchy as the relative hierarchy to the source node. For example, in Figure 4, Node A's hierarchy is $H(A, sourcenode) = 0$; Attribute E comes from higher hierarchy and its parent's node is located at one level higher than the source node, so it is defined as $H(E, sourcenode) = 1$; Attribute D located at lower hierarchy is $H(D, sourcenode) = -1$.

3.1.2 Ranking score definition

According to the hierarchical relationship in the graph, if we want to find the target information with given information, directly connected information item and its superclass's information item should be ranked higher; Others' ranks are determined by their structural position on the graph. Firstly, we introduce our final ranking score, which considers the effectiveness of all source nodes to the target node. Given a query graph $Q(V_s, t)$ and a specific $node\ u$ with type t, the ranking score is:

$$R(u) = \sum_{v \in V_s} r(u, v)$$

Where V_s is the set of source nodes in the query graph $Q(V_s, t)$ and $r(u, v)$ is the closeness score of $node\ v$ and $node\ u$. For single source's query, the ranking score is simplifies as $R(u) = r(u, v)$ when $u \neq v$. For star query, the ranking score is the summation of closeness scores as above; every closeness score is between $node\ u$ to each source node.

The closeness score aims at evaluating the closeness between two nodes based on both distance and hierarchy information, so it should meet following requirements: a. The closer two nodes are, the higher closeness score is; b. More shortest paths between two nodes make the closeness score higher; c. When we takes hierarchy changes into consideration, different hierarchy changes have different influences on the closeness score and results should follow the hierarchical relationship on the graph. Based on previous requirements, the closeness score is defined as:

$$r(u,v)=\begin{cases} 1 & \mathit{if}\ u=v \\ \alpha^{\max\{l_{(u,v)}-H_{\max}(u,v)-1,\ \frac{l_{(u,v)}-H_{\max}(u,v)}{N(u,v)}\}} & \mathit{otherwise} \end{cases}$$

Where $l_{(u,v)}$ is the length of shortest path between *u* and *v,* $N(u,v)$ is the number of shortest paths from $node\ v$ to $node\ u$, α is a constant between 0 and 1; $H_{\max}(u,v)$ is the maximum relative hierarchy between u and v, and it has direction property, so $H_{\max}(u,v)=-H_{\max}(v,u)$. The reason why we use maximum relative hierarchy between u and v is that we want to make good use of hierarchical information, and if we make sure a node located at higher level and have higher score, we have no reason to rank it lower. Therefore, we can make sure that the attribute of known data and attributes that belong to its superclass have largest closeness score 1. For other results, their closeness scores are always smaller than 1 and depend on $N(u,v)$, $l_{(u,v)}$ and $H_{\max}(u,v)$. When $u \neq v$, $l_{(u,v)}-H_{\max}(u,v)$ shows the deviation away from results with higher ranking scores on the graph and is always no smaller than 0; Larger deviation makes ranking score smaller, which also meets hierarchical structure of the graph. For non-hierarchical relationship on the graph, we just need to set $H_{\max}(u,v)$ to be 0 in the closeness score.

If $node\ u$ is located at 100 hops away from $node\ v$ and $node\ w$ is located at 101 hops away from $node\ v$, their ranking score should be similar as ratio of distance is 0.99 which indicates their distance differences can almost be ignored. But if $node\ u$ is located at 1 hop away from $node\ v$ and $node\ w$ is located at 2 hops away from $node\ v$, the ratio is 0.5 and the difference of distance is very important in evaluating the ranking score. So we design our ranking score in exponential form, which fits ranking score attributes mentioned above.

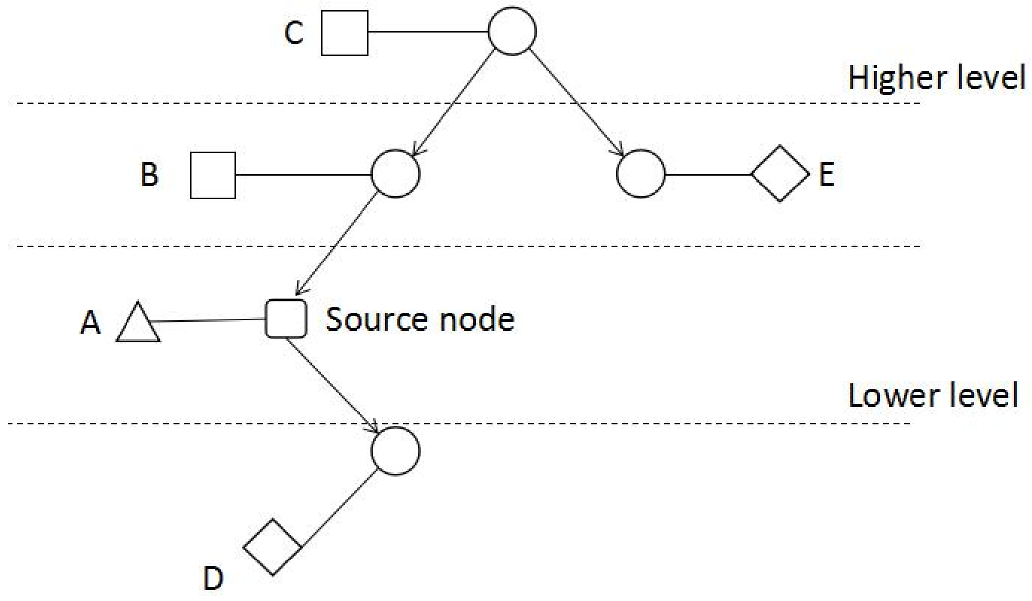


Figure 4 An example of knowledge subgraph with hierarchical structure

3.2 Bounding ranking score

To find *top-k* answers, a naive way to get them is to list out all nodes with the type we want, and sort them by their ranking score, then we can get *top-k* answers from all sorted answers. In order to accelerate the process of identifying *top-k* answers, we use the bounding ranking score of each node to terminate searching on some parts of graphs and filter out impossible results.

As is mentioned above, we use bounding ranking scores, instead of exact ranking scores, to find *top-k* answers. So for a specific target node *u*, we define the bounds of ranking score as:

$$\underline{R^t(u,V_s)} = \sum_{s \in V_s} \underline{r^t(u,s)}$$

$$\overline{R^t(u,V_s)} = \sum_{s \in V_s} \overline{r^t(u,s)}$$

Where V_s is the set of source nodes, $\underline{r^t(u,s)}$ and $\overline{r^t(u,s)}$ are the lower and upper bounding of the closeness score. The reason of using the sum of ranking score is to add all influence from source nodes to our final metric and show the overall effectiveness. The definition of bounding of closeness score is discussed below.

Given a source node, at each iteration we refines the bounding closeness score, and update the bound of *top-k* lower-bound closeness scores. Then we use the kth largest lower-bound ranking score to do termination check on all paths and terminate searching on paths whose upper-bound ranking scores have already smaller than the kth largest lower-bound ranking score. Following is the illustration of ranking score's upper and lower bounds.

We denote the source node as s and other nodes as v. The upper-bound and lower-bound closeness score of s and v is denoted as $\underline{r^t(v,s)}$ and $\overline{r^t(v,s)}$, where t represents the iteration number of query. Initially, when $t=0$, the upper-bound and lower-bound of the source node are:

$$\underline{r^0(s,s)} = 1 \text{ and } \overline{r^0(s,s)} = 1$$

For other nodes, which are at least one step away from the source node, their bounding closeness scores are set to be:

$$\underline{r^0(v,s)} = 0 \text{ and } \overline{r^0(v,s)} = \alpha^{-H_{\max}(u,v)}$$

At each iteration, the lower-bound ranking score of node v is updated using all its neighbors. In the following formula, $S(v)$ is the set of parent nodes of v, and we use its parent nodes' lower bound to get v's upper-bound.

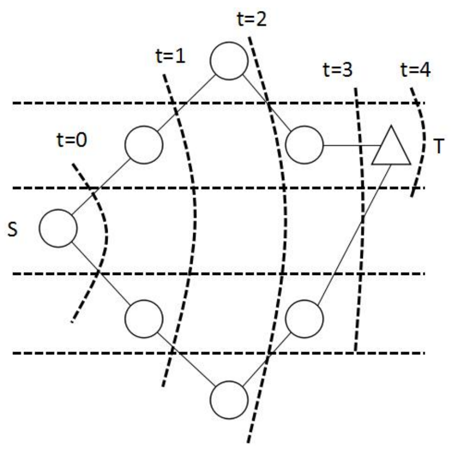
$$\underline{r^t(s,v)} = \begin{cases} \underline{r^{t-1}(v,s)} & \underline{r^{t-1}(v,s)} \neq 0 \\ \min\{\alpha^{t-H_{\max}(v,s)-1}, (\alpha \bullet \max_{n \in S(v)}\{\underline{r^{t-1}(n,s)} \bullet \alpha^{H(v,n)}\})^{\frac{1}{|S(v)|}}\} & \underline{r^{t-1}(v,s)} = 0 \end{cases}$$

For upper-bound of node v, if its lower bound at $t-1$ iteration remains zero , that means this node hasn't been visited during the query. So its upper-bound should keep decreasing, and may remove itself from candidate set, or stopping searching on this path, here we set it to be $\alpha^{l_{(u,v)}-H(v,s)-1}$. Otherwise, its upper bound should be set as larger one between its lower bound $\underline{R^t(v,s)}$ and $\alpha \bullet \overline{r^{t-1}(v,s)}$, because we want to let each node on the graph can be considered as much as possible. So we define it as:

$$\overline{r^t(v,s)} = \begin{cases} \max\{\underline{r^t(v,s)}, \alpha \bullet \overline{r^{t-1}(v,s)}\} & \underline{R^t(v,s)} \neq 0 \\ \alpha^{l_{(v,s)}-H(v,s)-1} & \underline{R^t(v,s)} = 0 \end{cases}$$

For those nodes who are visited more than twice in different iterations, we always ignore later visits, because they should have smaller ranking score and plays minor impact on ranking.

In Figure 5, we gives an example of single source query and its changes of ranking scores.



Iteration t	level	$\overline{R(T,S)}$	$\underline{R(T,S)}$
0	0	α^{-1}	0
1	0	α	0
2	0	α^{2}	0
3	0	α^{3}	0
4	$\max\{1,-1\}$	$\max\{\alpha^{4},\alpha^{\frac{3}{2}}\}$	$\alpha^{\frac{3}{2}}$

Figure 5 An example of ranking score bounds propagation

3.3 Top-k Emergence Test

In this subsection, we illustrate the *top-k* emergence test which can help us determine if the *top-k* answers have been found. During the process of searching results, we can get intermediate *top-k* lower-bound ranking scores of found results. For still visiting nodes, we can compare the upper-bound ranking score of currently visiting nodes $\overline{R_i}$ to kth largest lower-bound $\underline{R}_k$ of found results, and then terminate paths whose $\overline{R_i}$ smaller than $\underline{R}_k$. The reason of stopping searching on those paths is that the defined the ranking score function is monotone decreasing, and even if there is preferred results on this path, they will not have influence on final results.

In Figure 6, we give the pseudo-code of our query process. Given the query graph $Q(V_s,t)$, we start searching from all the source nodes, and find all nodes with type t as candidate set; During this process, we need to refine upper and lower bounds of candidate set, and do termination check. After several iterations, we can get a candidate set ψ with size k and return it.

```
Input: Graph $G(V_G, E_G, l_G(V), l_G(E))$ ,Qeury graph $Q(V_s, t)$ , Candidate size k
Output: Top-k canddiates set $\psi$

Procedure of querying:
    Compute $\overline{R^t(v,s)}$ for currently visiting nodes based on all sources
    Termination check:
        terminate query if $\overline{R^t(v,s)}$ is smaller than kth largest
        $\underline{R^t(s,v)}$
    update top-k $\underline{R^t(s,v)}$ scores
    keep record of top-k answers
    if $\psi$'s size = k
        return $\psi$
    else
        continue querying
```

Figure 6 Pseudo-code of implementation

3.4 Breadth-first search to find candidates

Starting from a source node, we use the Bread-first search(BFS) to find target nodes. Along all paths from the source node to target nodes, we have two condition to terminate searching. One is that nodes are the type that we want, and use its ranking score to do further processing like bounds refinement; The other is to use bounds of ranking scores. An example BFS process can be seen in Figure 7. In this process, we start searching at Node A. When Node F is found, and its type meets our requirement and it will be saved for further processing. As BFS goes, we can find Node B and its upper-bound ranking score isn't enough, then we will terminate searching nodes along this path, even if there are nodes with type we need. In the next subsection, we show the distributed strategy using this branch and bound algorithm to improve its query performance.

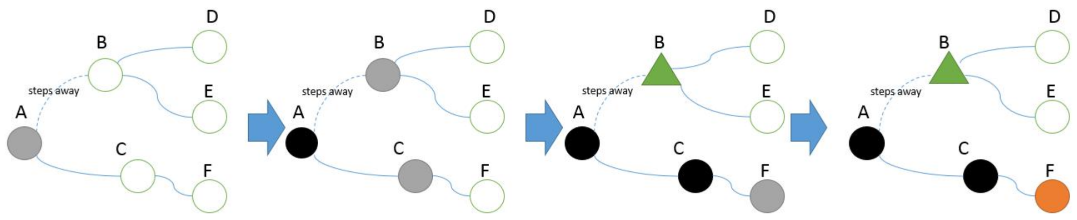


Figure 7 BFS process

CHAPTER 4

DISTRIBUTED IMPLEMENTATION

4.1 Distributed Implementation

To show the previous algorithm's distributed implementation strategy, our algorithm is implemented on the Hadoop [22], a popular distributed computing framework. Hadoop includes two parts, one is map-reduce computing paradigm, which computes the ranking algorithm; the other is HDFS, which helps us to save intermediate values between two iterations.

4.1.1 Breadth-first search on Map-Reduce

In order to implement Breadth-first search using map-reduce paradigm, we need different labels to show the process of searching. For nodes which are being visited during the search, we label them as gray color; For nodes haven't been visited, we mark them as white; Nodes have been visited, we mark them as black; When we find target type of nodes, mark them as red. Each time, we only generate new nodes who are neighbors of processing nodes(gray nodes), and use these new nodes to update the status of these neighbors, both label(color) and ranking score are managed in reduce phase. Because we still need to record distance during between the source node and target nodes, and can't let each path vacillate back and forth between two nodes, each time generating new nodes based on neighbor, we should ignore parent nodes. In other words, the status of parent nodes can't be updated by their son nodes.

To implement on the Hadoop, after each iteration, we need to save current states of all nodes into HDFS. For the next iteration, computing framework needs to read them from HDFS for computation of current iteration. The Hadoop process in Figure 7 is shown in Figure 8.

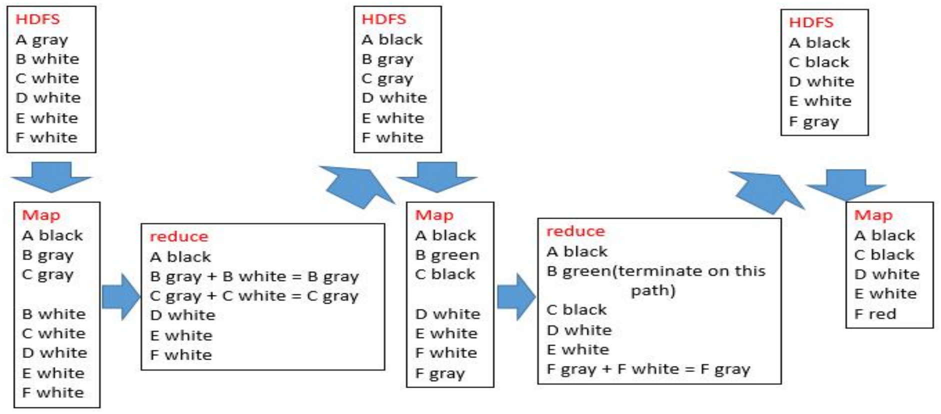


Figure 8 BFS process on Hadoop

4.1.2 Top-k Identification on Distributed framework

After each iteration of breadth-first Search, some nodes are labeled and ranking scores are computed. Then the results of search are selected and saved in the HDFS.

As talked about in the previous section, in order to accelerate finding results, we need to get a filtered candidate set and terminate paths that are impossible to find candidate nodes. In Figure 9, it shows the procedure of data processing in each iteration. Each time, data is loaded from HDFS to the master node of the cluster; In the master, each record is pre-processed by appending *kth* largest lower bound ranking score found in the previous iterations; Then records are assigned to workers for further computation. In each worker, two jobs are needed. Firstly, calculating its own ranking score based on its parent's ranking score and the distance away from the source node; After that, we compare the ranking score to the *kth* largest one and determine if we need to continue searching on this path; If not, just set as visited, otherwise, set it to be visited and created new records of its neighbors for the other job. Secondly, based on output of previous jobs, update the status of nodes found by previous jobs. Before saving the processed data into HDFS, we need to update the *top-k* largest lower bound ranking score if target nodes are founded in master machine.

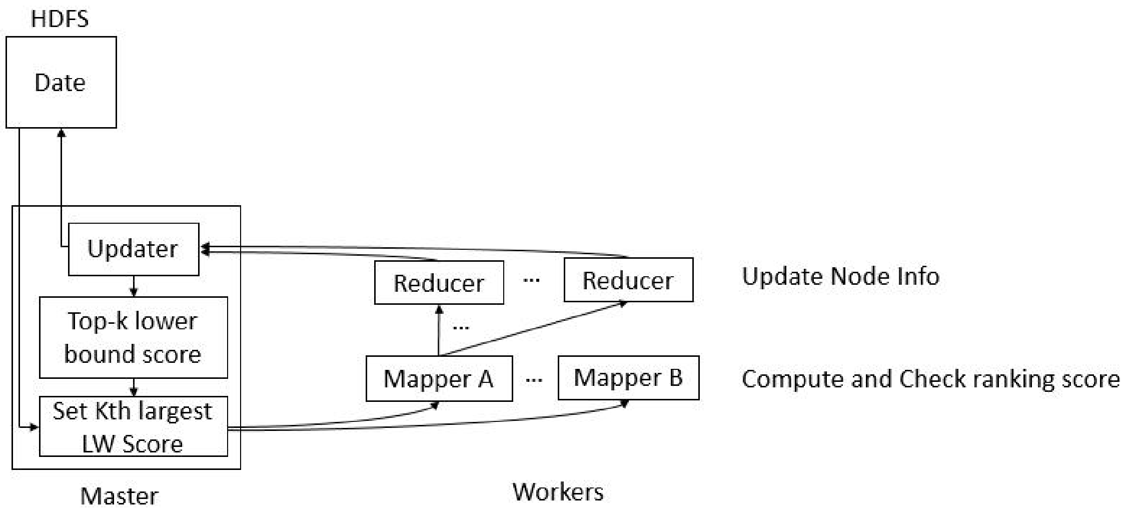


Figure 9 Distributed implementation structure

To sum up, the way to terminate searching on a specific path is to stop updating the status of its neighbors, meanwhile lots of works about networking and updating status can be removed, like Node B in Figure 7 and Figure 8. *Top-k* lower-bound ranking score can also be listed out during BFS, which saves time of sorting lower-bound ranking scores and getting the *kth* largest one on single machine.

CHAPTER 5

EXPERIMENTAL EVALUATION

In this chapter, we present our graph analysis to prove the significance and necessity of querying on hierarchical knowledge graphs. Then we evaluate the performance of our ranking algorithm by comparing with baseline algorithm's results.

5.1 Experimental dataset

The dataset we use to test our algorithm and implementation has shown as the example in first section, and the schema is shown in Figure 1. We collect these data from nearly 2000 CISCO official web pages, the majority of data is about CISCO products and its attributes. The information is extracted by web crawler implemented by dom4j, including both structured entities, semi-structured entities and plain text containing useful content. More graph details are listed in table 1.

Number of node types	Number of edge types	Number of nodes	Number of edges	Average Degree
3	2	8978	24990	5.57

Table 1 Graph specific information

Semi-structured data refers to information that can be extracted directly from web pages without much post processing. For instance, BugId is located at introduction part, we just reach to this part of HTML source and target at specific type of tags or ids, then we can get the bug id. Because this bug is also related to certain vulnerability, we store this relationship as well. Meanwhile, information in the tables and bulleted lists is also extracted in the similar way.

Obviously, Structured data can be easily used to build the hierarchical knowledge graph, as the relationship among items has already defined in the web pages. But for plain texts, we can't take a paragraph as an item, so further processing is still needed. To identify and extract information segment from a paragraph, we use natural language processing(NLP) algorithm, Part-of-Speech Tagger [19]. Most recent Part-of Speech Tagger uses Conditional Random Field [16] to tag word's sequences and find entities in our plaintext.

To formulate information items into hierarchical knowledge graph, more steps are needed. First, we use k-means clustering [10] to group information items, so similar products are classified into one group. Products in each group represent tightly-related information items, for instance, same products with different versions. In this way, we get various groups. In each group, hierarchy is built based on facts of their relations, for example, some information items are subset of others. Among groups, they are connected based on similar attributes or common technologies. Compare to relevance of items inside each group, relationships of different groups are not that close, so these connections should locate at lower level in the hierarchical knowledge graph.
The work flow of constructing hierarchical knowledge graph is shown in Figure 10.

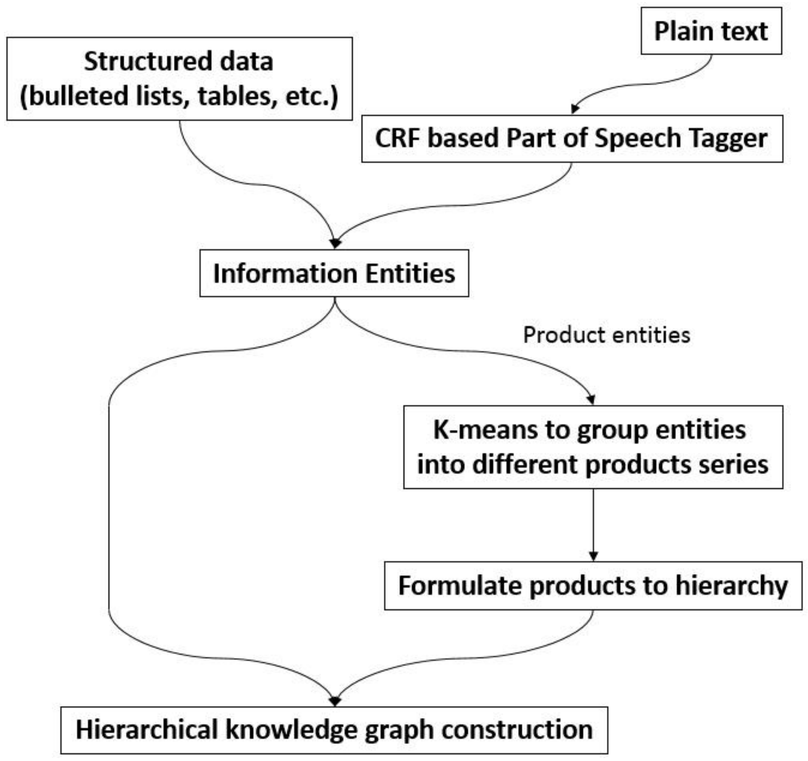

Plain text
Structured data (bulleted lists, tables, etc.)
CRF based Part of Speech Tagger
Information Entities
Product entities
K-means to group entities into different products series
Formulate products to hierarchy
Hierarchical knowledge graph construction

Figure 10 Work flow of constructing KB graph

5.2 Viability for hierarchical structure analysis

The key purpose of using hierarchy information in knowledge graphs is to find a complete query result set for users. In the knowledge graph with hierarchical relationship, if a node v doesn't have any properties, but directly or indirectly connected to other nodes with properties, these properties can potential query results; Or node v have lots of meaningful properties, it maybe helpful to increase the size of its related nodes' information set.

Because the hierarchical structure plays an important role in our query results, we use following metrics to evaluate our hierarchical structure in the knowledge graph:

X is the number of attribute nodes with designated property connected to a CISCO's product *node v* ;

Y is the number of designated attribute nodes connected to node *v*'s same hierarchy or higher hierarchy nodes;

Z is the number of CISCO's product nodes with same or higher hierarchy of *node v* and designated properties.

Z/X ratio is the ratio of higher level products nodes and designated connected nodes to the source node. We use Z/X ratio to show the complexity of hierarchical structure; The larger than Z/X ratio is, the more complex the hierarchical structure is. And Y/X ratio indicates that we can find more potential results using hierarchical relationships than directly connected results. We use these two metrics to show the necessity and viability of our query in hierarchical knowledge graphs, which can help us find more interesting results. We randomly choose 20 CISCO product in our graph and use Z/X ratio and Y/X ratio to evaluate properties of our graph. Z/X ratio indicates the number of related nodes in the hierarchical structure, which may also shows the necessity to do search with hierarchical relation information. Y/X ratio shows how many potential results may be found based on hierarchical knowledge graph. In Figure 3(b), for product *Cisco Unified IP Phones 9900 Series version 9.3(.4) and prior*, its X value is 1, Y value is 5 and Z value is 3; So Z/X ratio is 3 and Y/X ratio is 5, and these two values show that complex hierarchical structure related to this product; We can also potentially get more trustworthy results based on hierarchical relationships.

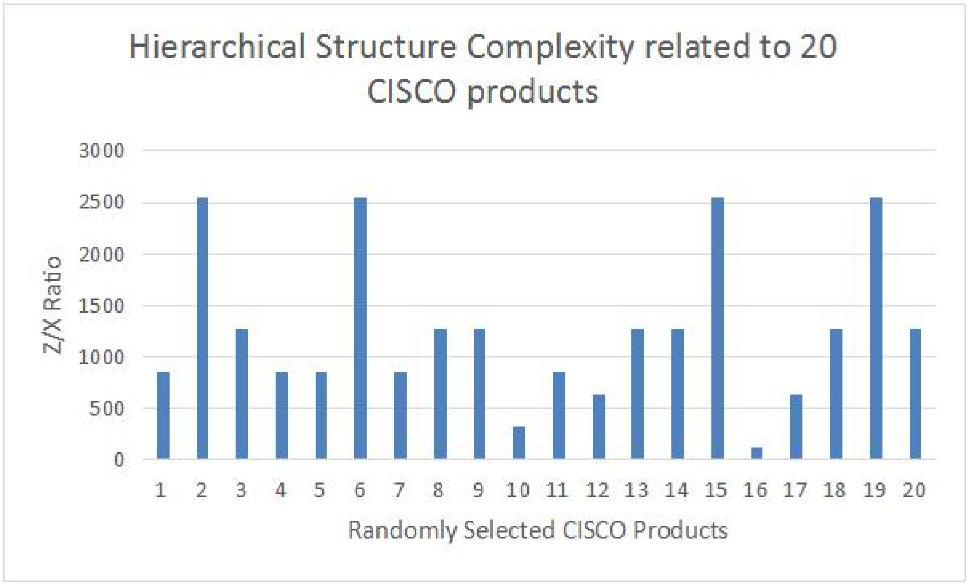


Figure 11 Hierarchical structure complexity of sample products

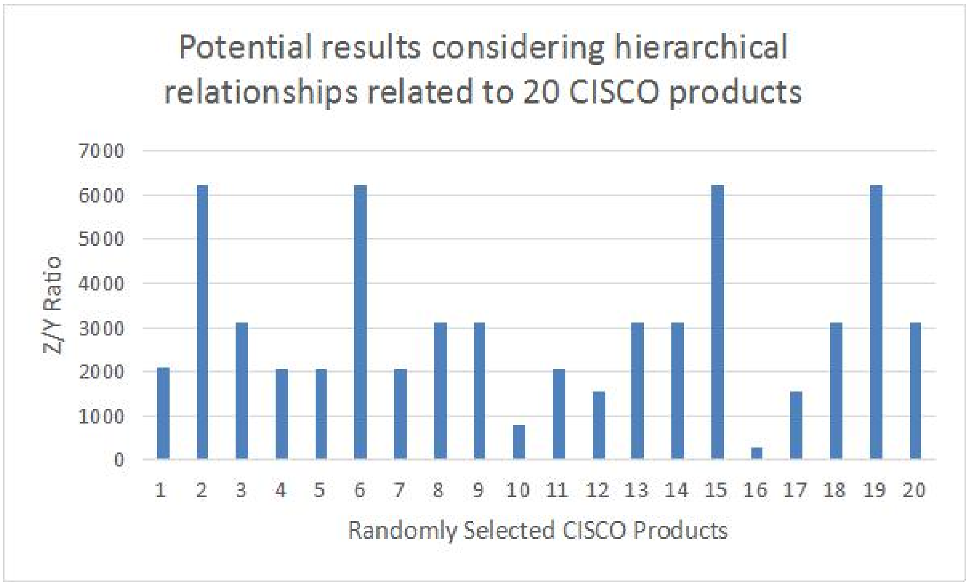


Figure 12 Potential results considering hierarchical structure

Using these two metrics, we can see complex hierarchy structure and a great number of potential results can be found in hierarchical knowledge graphs. It shows our query is meaningful and useful to optimize current graph search concepts and algorithms.

5.3 Single source's query example and analysis

The section focuses on our current evaluation work on the accuracy of the query. All experiments are operated on virtual machines deployed on VMware workstation. The specification of each machine is single core and 1GB memory running CentOS Linux Version 6. The algorithm is implemented on Hadoop-1.2.1, and parameter $\alpha = 0.5$. Currently, the number of nodes extracted from CISCO official websites is about 10000. We use the CISCO product information dataset to test our algorithm and implementation. Following is the discussion on the accuracy of *top-k* answers.

In the CISCO products hierarchical knowledge graph, we present following query example in Figure 13.

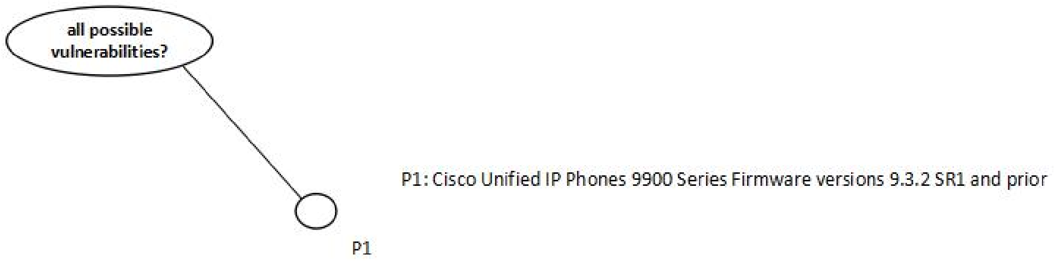

all possible vulnerabilities?
P1: Cisco Unified IP Phones 9900 Series Firmware versions 9.3.2 SR1 and prior
P1

Figure 13 Single source's query example on the CISCO product graph

The example in Figure 13 is intended to find *top-5* vulnerabilities that affect Cisco *Unified IP Phones 9900 Series Firmware versions 9.3.2 SR1 and prior*. In Figure 14, we present its corresponding subgraph and node-44 is its id. Apparently, not only vulnerabilities with id node-3184, node-5593 and node-3263 are connected to this product, a lot more related vulnerabilities can be seen on the subgraph. In this graph:

Blue Nodes: CISCO Product

Red Nodes: Vulnerability

Green Nodes: Bug_Ids

Yellow Nodes: Workaround

And our source node is Node 44(largest one) here.

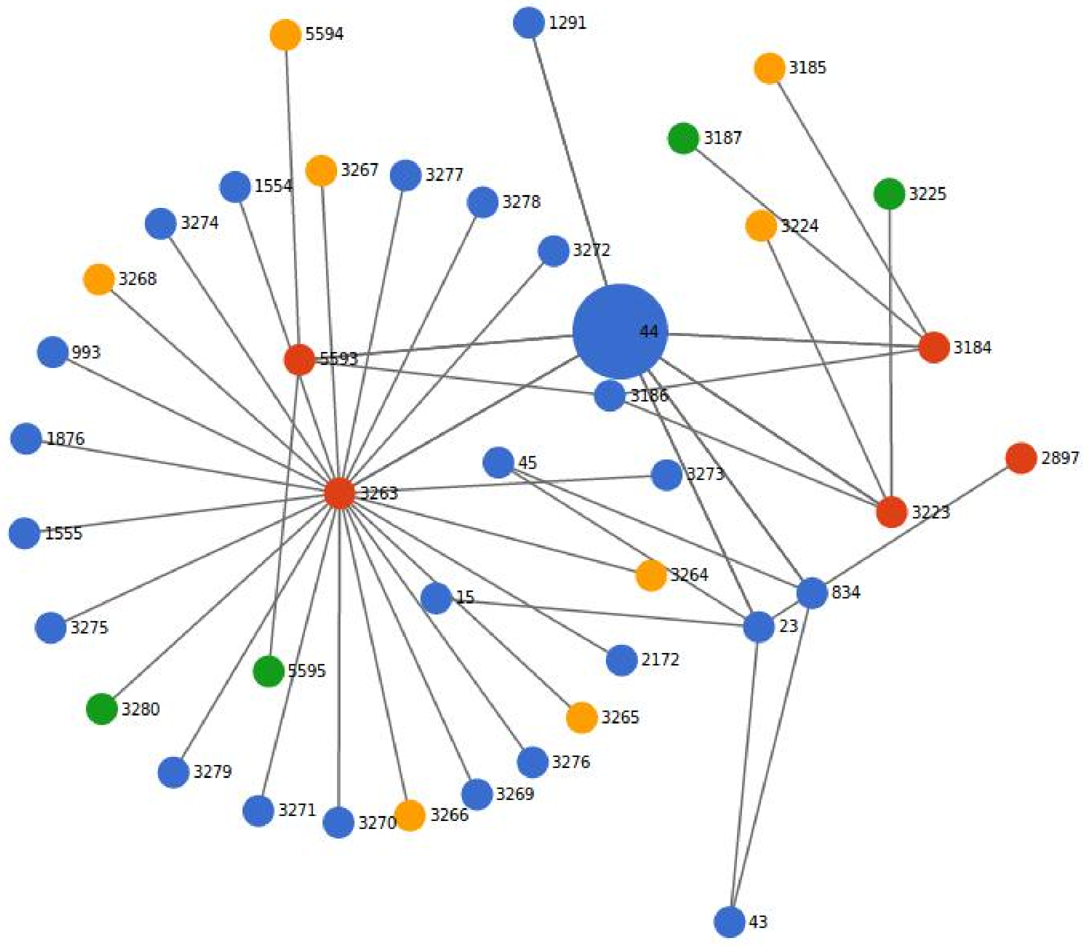


Figure 14 A subgraph example in CISCO product hierarchical knowledge graph

When α is 0.8, the *top-5* query results are:

Node ID	Node Name	Ranking Score
3184	cisco 9900 series phone webapp buffer overflow Vulnerability	0.8
3223	cisco unified ip phones 9900 series image upgrade command injection vulnerability	0.8
3263	multiple vulnerabilities in cisco ios xe software for 1000 series aggregation services routers	0.8

5593	cisco unified ip phone 8900/9900 series crafted sdp packet vulnerability	0.8
2897	cisco 9900 series ip phone crafted header unregister vulnerability	0.64

Table 2 Query results of single source

According to the semantic meaning, we can see our results are effective and find a potential vulnerability for this product. The query results with higher ranking scoress are directly connected to the node-44; As for one with lower ranking score, it belongs to node-44's subclass, and we take it as potential results.

In order to show the effectiveness of querying on hierarchical knowledge graphs, we compare our results with others. Here, we use results got from traditional knowledge graph considering only the distance as metrics and compare with our hierarchy-based query results, then we also identify if they are correct by providing related ground truth. In Figure 15, we randomly choose 8 products and find *top-10* related vulnerabilities on both traditional knowledge graphs and hierarchical knowledge graphs, and compare their query results. In the figure, we can always find some different query results. In order to evaluate our query results, we search related materials of each product in the figure and use the information as ground truth to evaluate our results. Based on our check with ground truth, we can make sure our query results are more reliable and useful.

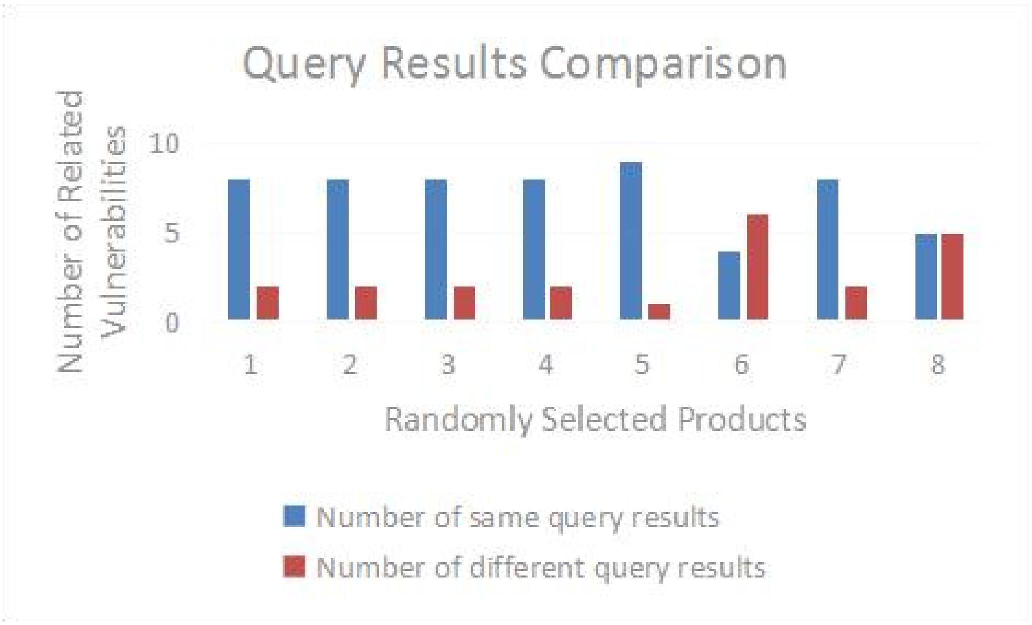


Figure 15 Query results comparison

5.3 Star query example and analysis

If we have more and corresponding nodes on the graphs, we can also use our algorithm to get useful results. Figure 16 is an example to search on the hierarchical knowledge graph, we want to find common vulnerabilities of *cisco asa software versions prior to 9.0(3.8)*, *cisco telepresence system software versions ix 8 (.0.1) and prior*, *cisco asa software releases 9.3(.2)* and *prior and cisco asa versions 9.1(.3) and prior*. Three of these four products belong to a same series; The other one is a different product, it aims to show the effectiveness to use graph relationships to find potential interesting results.

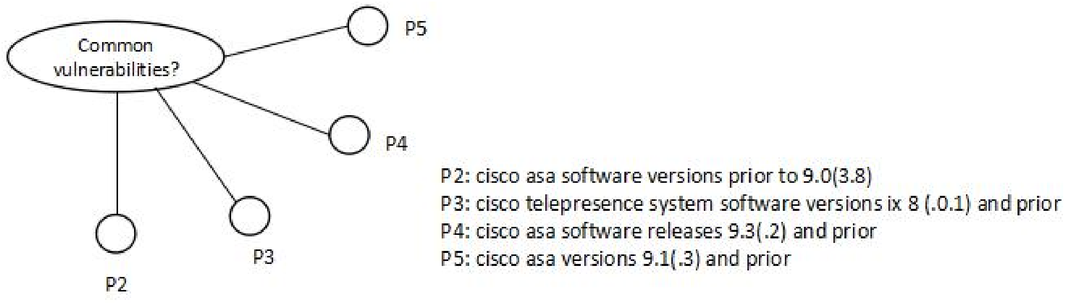

Common vulnerabilities?
P5
P4
P3
P2
P2: cisco asa software versions prior to 9.0(3.8)
P3: cisco telepresence system software versions ix 8 (.0.1) and prior
P4: cisco asa software releases 9.3(.2) and prior
P5: cisco asa versions 9.1(.3) and prior

Figure 16 star query example on the CISCO product graph

When α is 0.8, the query results of our algorithm and implementation are listed below:

Node ID	Node Name	Ranking Score
3115	cisco adaptive security appliance ssl vpn authentication bypass vulnerability	3.4
4409	gnu bash environment variable command injection vulnerability	2.314
4750	multiple vulnerabilities in openssl affecting cisco products	2.168
6183	cisco asa clientless ssl vpn portal customization integrity vulnerability	2.0
7713	multiple cisco products root shell access vulnerability	1.952
6179	cisco asa authenticated linux shell access vulnerability	1.928
4630	multiple vulnerabilities in ntpd affecting cisco products	1.828
6217	cisco asa smart call home digital certificate validation vulnerability	1.8
6638	openssl heartbeat extension vulnerability in multiple cisco products	1.8
7315	cisco small business rv series and sa500 series dual wan vpn router generated key pair information disclosure vulnerability	1.8

Table 3 Query results of star query

Here, vulnerabilities with higher ranking score are more likely to be common vulnerabilities. Compared with baseline algorithm mentioned before, our query answers are located at different positions. To evaluate the relative positions of query answers to sources, we use following metric:

$$\eta = \frac{\sum_{v \in V_s} dist(u,v)}{|V_s|}$$

Where V_s is the set of all source nodes, u is a answer and $dist(v,u)$ represents the shortest distance between v and u.

If we only consider distance as the factor to find vulnerabilities, the potential answers should be located at the middle among all source nodes and their η are 5 on average. But for the first answer of our query algorithm - *cisco adaptive security appliance ssl vpn authentication bypass vulnerability,* its η is 3.75. Apparently, our query results are quiet different.

Finally we use ground truth of knowledge to evaluate our query results. For the result *cisco adaptive security appliance ssl vpn authentication bypass vulnerability,* it directly connects to P2 and is located at higher level relative to P3, P4 and P5. According to semantic meanings and ground truth we get on CISCO's official websites, this result is good one and very trustworthy.

CHAPTER 6

RELATED WORK

In the field of graph data management, a lot of explorations have been made in recent years. These algorithms are applied in social networks, bioinformatics, and software engineering projects. It becomes nontrivial to manage graphs in different ways. Most of graph management researches focus on five major topics, which are graph queries, graph labeling strategies, *top-k* identification, distributed graph queries and semantic-based query. In the following part, we introduce them respectively.

6.1 Graph Queries

Most of graph queries put emphasis on pattern search [24, 13] and reachability [3, 15]. As for first kind, answers are got from subgraph isomorphism. Isomorphic subgraph may not be exactly same as subgraphs, so they use missing nodes or edges as ranking metrics to eliminate unconfident candidates [21] or show the factor of missing nodes and edges in final score [12]. Some other works focus on graph indexing[25] to improve the performance of queries. Two major categories of graph index: (1) Non mining-based graph indexing techniques[26, 27]: this technique is to index the whole graph; (2) Mining-based graph indexing techniques[28, 29]: instead of indexing the whole graph, this technique extracts features from the graph and use these features to get inverted index; When querying a subgraph, we need to extract features of this subgraph first and then use inverted index to find it on the whole graph. In our work, we use index-free algorithm and propagate the ranking score to our target nodes.

6.2 Graph Labeling Strategies

The graph labeling is mainly for nodes in the graph. The goal of indexing these nodes always aims to help find isomorphic subgraphs [14] or accelerating query speed [4, 7]. The first object has illustrated before. To decrease the query time, more information of nodes and their neighbors should be stored; And then some low cost indexing algorithms are also presented [4]. In our work, we not only label nodes with different types, but we also label edge to show meaningful relations. Labels in this paper don't have too much function to increase query efficiency.

6.3 Top-k Identification

To get *top-k* answers from candidate set, there are two main algorithms. First one is the baseline method, which sort all candidates and then get k best answer from sorted outcomes. But this way costs more memory and time. The other way is based on branch and bound algorithms [6]. This thought is to keep refining bound of score to terminate part of operation for optimization, then get t*op-k* candidates directly. Here, we use this concept to design our ranking score and make it compatible to distributed system. In this way, efficiency and computation resources are improved.

6.4 Distributed Graph Queries

Many distributed frameworks are proposed, such as Hadoop [20], GraphLab [17], etc. These frameworks connect different servers for parallel computation. They combine and regulate computation and storage resources for good performance. Graph queries can also be designed on distributed framework to achieve higher performance [11] and scalable objects [23]. In this paper, we use Hadoop to implement more fast branch and bound algorithms, then find *top-k* answers.

6.5 Semantic Parsing Query

Using semantic parsing [22, 18] on famous knowledge bases like Freebase [2], helps us deeply understand knowledge bases. Related research uses semantic parser to map natural language into logical format that can be executed on knowledge graph, and then gives us more meaningful outcomes from graph query. In this paper, we learn from the essential thought of semantic parsing, and use semantic relation to build the knowledge graph, which demonstrates showing authentic results collaborating our query algorithm.

CHAPTER 7

CONCLUSION

In our work, we use both hierarchical relationship and non-hierarchical relationship in the knowledge graphs to do graph query. Especially, we use hierarchical relationship to find more interesting results which can not obtained by traditional search engines and other graph search algorithms. We also propose *top-k* query algorithm to do fast query on knowledge graphs with hierarchical relationships. The algorithm uses bounding ranking scores and found results to do termination check, then we can query on part of the knowledge graph to find interesting results. An implementation on the distributed framework is proposed, in case that we need to query on much larger scale knowledge graphs. Finally, we test the algorithm at two situations, single source node query and star query. Using ground truth related to our dataset, the experiments demonstrate that our query answers are more trustworthy and interesting.

BIBLIOGRAPHY

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. Springer, 2007.

[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1247-1250. ACM, 2008.

[3] E. P. Chan and H. Lim. Optimization and evaluation of shortest path queries. The VLDB Journal, 16(3):343-369, 2007.

[4] J. Cheng, Y. Ke, A. W.-C. Fu, and J. X. Yu. Fast graph query processing with a low-cost index. The VLDB Journal, 20(4):521-539, 2011.

[5] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang. Fast graph pattern matching. In Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on, pages 913-922. IEEE, 2008.

[6] J. Clausen. Branch and bound algorithms-principles and examples. Department of Computer Science, University of Copenhagen, pages 1-30, 1999.

[7] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. SIAM Journal on Computing, 32(5):1338-1355, 2003.

[8] Y. Duan, X. Li, and L. Ding. A reachability query method based on labeling index on large-scale graphs. In 2015 International Conference on Computational Science and Computational Intelligence (CSCI), pages 77-82. IEEE, 2015.

[9] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, pages 156-165. Society for Industrial and Applied Mathematics, 2005.
[10] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1):100-108, 1979.
[11] B. Izs_o, G. Sz_arnyas, I. R_ath, and D. Varr_o. Incquery-d: Incremental graph search in the cloud. In Proceedings of the Workshop on Scalability in Model Driven Engineering, page 4. ACM, 2013.
[12] J. Jin, S. Khemmarat, L. Gao, and J. Luo. A distributed approach for top-k star queries on massive information networks. In Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on, pages 9-16. IEEE, 2014.
[13] I. Jonassen. E_cient discovery of conserved patterns using a pattern graph. Computer applications in the biosciences: CABIOS, 13(5):509-522, 1997.
[14] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. Nema: Fast graph search with label similarity. In Proceedings of the VLDB Endowment, volume 6, pages 181-192. VLDB Endowment, 2013.
[15] S. Khemmarat and L. Gao. Fast top-k path-based relevance query on massive graphs. IEEE Transactions on Knowledge and Data Engineering, 28(5):1189-1202, 2016.
[16] J. La_erty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings of the eighteenth international conference on machine learning, ICML, volume 1, pages 282-289, 2001.

[17] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. Proceedings of the VLDB Endowment, 5(8):716-727, 2012.

[18] S. Reddy, M. Lapata, and M. Steedman. Large-scale semantic parsing without question-answer pairs. Transactions of the Association for Computational Linguistics, 2:377{392, 2014.

[19] K. Toutanova and C. D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13, pages 63-70. Association for Computational Linguistics, 2000.

[20] T. White. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012.

[21] D. W. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In 2007 IEEE 23rd International Conference on Data Engineering, pages 976-985. IEEE, 2007.

[22] W.-t. Yih, M.-W. Chang, X. He, and J. Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In Association for Computational Linguistics (ACL), 2015.

[23] H.-F. Yu, C.-J. Hsieh, H. Yun, S. Vishwanathan, and I. S. Dhillon. A scalable asynchronous distributed algorithm for topic modeling. In Proceedings of the 24th International Conference on World Wide Web, pages 1340-1350. ACM, 2015.

[24] L. Zou, L. Chen, and M. T.   Ozsu. Distance-join: Pattern match query in a large graph database. Proceedings of the VLDB Endowment, 2(1):886-897, 2009.

[25] Sakr S, Al-Naymat G. Graph indexing and querying: a review[J]. International Journal of Web Information Systems, 2010, 6(2): 101-120.
[26] Sakr S. GraphREL: A decomposition-based and selectivity-aware relational framework for processing sub-graph queries[C]//International Conference on Database Systems for Advanced Applications. Springer Berlin Heidelberg, 2009: 123-137.
[27] Jiang H, Wang H, Philip S Y, et al. Gstring: A novel approach for efficient search in graph databases[C]//Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007: 566-575.
[28] Zhao P, Yu J X, Yu P S. Graph indexing: tree+ delta<= graph[C]//Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007: 938-949.
[29] Zhang S, Li J, Gao H, et al. A novel approach for efficient supergraph query processing on graph databases[C]//Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM, 2009: 204-215.