QUERYING ENTITY-RELATIONSHIP GRAPHS BY EXAMPLE TUPLES: EXPERIMENTAL EVALUATION AND USER STUDY

by

MAHESH GUPTA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2012

ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to my advisor Dr. Chengkai Li. Without his guidance and persistent help this thesis would not have been possible. I would also like to thank my committee members Dr. Leonidas Fegaras and Dr. Ramez Elmasri for being on my panel and giving me advice related to my thesis work.

Special thanks for Mr. Nandish Jayaram, with whom I worked in this project. He guided me in every step of this thesis. I would also like to thank Prof. Xifeng Yan and Mr. Arijit Khan from University of California, Santa Barbara who collaborated with us in this project.

I would like to express my gratitude to Center for Online Development at UT Arlington for funding me, providing all time support to me during my study and giving me opportunity to work on cutting edge technologies. Special thanks to my manager Mr. Duy Nguyen whose friendship I will always cherish in my life.

I owe my deepest gratitude to my parents for their constant inspiration, care and support throughout my life. I wish to thank my siblings for their inspiration and support.

November 23, 2012

ABSTRACT

QUERYING ENTITY-RELATIONSHIP GRAPHS

BY EXAMPLE TUPLES: EXPERIMENTAL

EVALUATION AND USER STUDY

Mahesh Gupta, M.S.

The University of Texas at Arlington, 2012

Supervising Professor: Chengkai Li

The World Wide Web today has evolved into a rich repository of entities where many knowledge bases containing entity-related information are directly available. Such knowledge bases are often in the form of entity-relationship graphs. To query entity-relationship graphs, users need to provide input entities, attributes and relationships by complex query graphs. To improve the usability of graph database systems, we study a novel mechanism that queries entity-relationship graphs by example tuples. It allows users to express a query in the form of one or more tuples consisting of entities. The underlying query system automatically builds a query graph based on the example tuples and ranks matching answer tuples.

The focus of this thesis is to evaluate our query system's accuracy and efficiency. To evaluate accuracy we employ two methods. In the first method we evaluate queries whose ground truths are known and calculate system's precision and recall. In the second method we conduct user study on ranked answer lists and calculate rank correlation co-efficient. The run time efficiency of the system is measured with respect to the size of the query graph.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The World Wide Web today has evolved into a rich repository of entities where many knowledge bases like Freebase [6] containing entity information are directly available. As of November 2012 Freebase dataset contains almost 23 million entities [8]. An entity is a single thing or concept that exists in the world. Entity-Relationship graph is a labeled graph where a node represents entity and edges are relationships between them. Compared to traditional keyword based search, entity-search targeting entities and their relationships can capture user intention better, hence generating more meaningful search results.

1.1 Motivation

To query an entity-relationship graph, a user has to either use a complex querying mechanism or provide a query graph as input. In both cases its very difficult for a novice user to query an entity-relationship graph. Jayaram et al. [31] proposed Graph Query by Example (GQBE) system for querying entity-relationship graph. GQBE provides solution for this problem. A user is expected to give example tuples as input to the system and the system automatically builds a query graph that is matched against the entity-relationship graph to find similar answer tuples. Since the query graph is automatically generated, it becomes imperative to measure the quality of the answer tuples obtained. So we need an experimental setup to measure the effectiveness of this new querying mechanism.

1.2 Overview of Problem

Suppose a user is interested in finding “a list of technology companies and its founders”. If she performs a keyword search on the Web, the result is a set of pages that might not have the list of company-founder pairs she is looking for. GQBE provides an alternative solution to this problem where the search is conducted on an entity-relationship graph. Query by example paradigm is easy to use like the keyword query paradigm even for a novice user. Here the user gives one or multiple answers to his query as example tuples to the system, and using this information, the system tries to find other similar answers and ranks them.

For instance, example tuple for the technology company and its founder list can be {Jerry Yang, Yahoo}, and the system is expected to find tuples like {Bill Gates, Microsoft}, {Larry Page, Google}, {Steve Jobs, Apple} etc. as the answers.

1.3 Challenges

Query by example imposes several challenges. The first challenge is to determine the user intention. For example Jerry Yang is not only the "founder" of Yahoo, but he is also a "board member" of Yahoo. Second challenge is to automatically and efficiently construct a query graph capturing important relationships and properties of the input entities. The third challenge, that is focus of this thesis work is, evaluating the correctness and quality of the answers.

1.4 Overview of Solution

GQBE automatically builds a query graph from the example tuple provided by the user. To ensure a small query graph, yet capturing important relationships, GQBE assigns weight to each edge in the graph using various intuitive parameters like Inverse Edge Frequency (IEF), frequency of an edge with respect to both its ends and the distance of an edge from input entities (describe in details in Chapter-3) to select "important" edges. Our experiments verify that this approach helps in capturing the user intention pretty well.

To check the correctness of the results, we designed input queries from gold set ground truths known to us. We select one of the tuples from a ground truth as the example tuple input to GQBE and then use the ground truth to calculate Precision-at-K [14], Normalized Discounted Cumulative Gain (nDCG) [16] and Mean Average Precision (MAP) [15] of the ranked list of answer tuples returned by GQBE. We also compared the precision and execution time of GQBE with another related system called NESS (Neighborhood based similarity search) [1]. An extensive user study using Amazon Mechanical Turk [26] was also conducted to measure the quality of the results returned by GQBE. We used a modified version of Pearson Correlation Coefficient [17] to capture the effectiveness of the ranking. More detailed description of the evaluation is presented in chapter-5.

1.5 Contribution of Thesis

In this thesis work, I have carried out an extensive experimental study of effectiveness and efficiency of GQBE on an entity-relationship graph. The main contributions of this thesis are:

- Freebase dataset cleaning and processing.
- Design of different queries and their ground truth collection.
- GQBE accuracy and efficiency comparison with NESS
- Measuring accuracy of the answers' ranking using Amazon Mechanical Turk user study.
- Developing an easy to use web interface for GQBE.

1.6 Organization of Thesis

The thesis is organized as follows. Chapter 2 gives a detailed background of other related work on querying entity-relationship graph. Chapter 3 presents the formal definition of concepts introduced by GQBE and different algorithmic approaches to find answers. Chapter 4 describes technical details of our implementation, preprocessing of Freebase dataset and the Web demo. Chapter 5 describes query design, ground truth collection, efficiency metrics, Amazon Mechanical Turk user study and results. Conclusion is drawn in Chapter 6. I have used material from our work [31] in this thesis.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we discuss similar work done and why they are not applicable to our problem.

Cohen et al. [3], Xin et al. [27] proposed technique for expanding set of named entities. The proposed system uses semi-structured web pages to find answers. GQBE is different from these on two main aspects – (1) while these systems find existing answers within structure in web pages such as HTML tables and lists, GQBE works on data graph. (2) GQBE is more general in that each input query tuple contains multiple heterogeneous entities whereas these systems, take multiple input objects in which each input is a single entity.

There are several works [25,28,29,30] that identify the best subgraphs/paths in a data graph to describe how several input nodes are related. The query graph discovery component of GQBE is different from these works in several important ways – (1) Some of these works focus on homogeneous graphs where all nodes are of the same type and all edges represent instances of the same relationship. GQBE focus on heterogeneous graphs with many different types of entities and relationships. (2) The graph identified by these works contains only those paths that connect the input nodes, but GQBE include relationship pertinent to individual query tuple entities. (3) GQBE uses query graph to find answer graphs and answer tuples, which is not within the focus of these works.

Tian et al. [2] proposed technique for approximate matching of large graph queries. The proposed method called TALE (a tool for approximate subgraph matching of large queries efficiently) uses a novel neighborhood indexing method for faster execution time. Khan et al. [1] proposed technique for efficient approximate search of a query graph in a noisy, large, incomplete, labeled data graph. The proposed method called NESS (Neighborhood bases similarity search) is appropriate for graphs with low automorphism and high noise.

GQBE is different from [1] and [2] on several aspects. First, GQBE requires to match only edge labels, but node labels are not required to be always matched. This is equivalent to matching a query graph with all unlabeled nodes, and thereby significantly increases the problem complexity. Second in GQBE query graph is weighted edges graph, and an answer graph must have entities corresponding to all query tuple entities. We compared GQBE with NESS and our empirical result shows that GQBE clearly beats NESS in accuracy. Finally, all these works assume query graph as input while GQBE automatically creates query graph from query tuple.

CHAPTER 3

OUR APPROACH

In this chapter we provide formal definitions of all term used in GQBE, construction of maximal query graph from user's example tuple, 2 different algorithmic approaches to find answer tuples and ranking methodology.

<u>3.1 Entity-Relationship Graph</u>

An entity-relationship graph is a labeled directed multi graph G (V, E) where V is a set of entities (vertices) and E is a set of directed relationship (edges) between the entities. We also use V(G) and E(G) to denote the entity set and the edge set, respectively. Each node $v \in V$ represents an entity. Each node has a unique identifier id. Each edge $e=(v_i, v_j) \in E$ denotes a direct relationship from entity v_i to entity v_j. Each edge has a label, denoted as label(e). Multiple edges can have the same label.

Figure 3.1 shows the example of an entity-relationship graph. Each node in the graph represents an entity and an edge represents the relationship between the two vertices. For instance vertex "Jerry Yang" represents an entity connected to another entity "Yahoo" by relationship *Founder*. Example of another relationship is *Industry*, where both "Google" and "Yahoo" connect to "Computer Software".

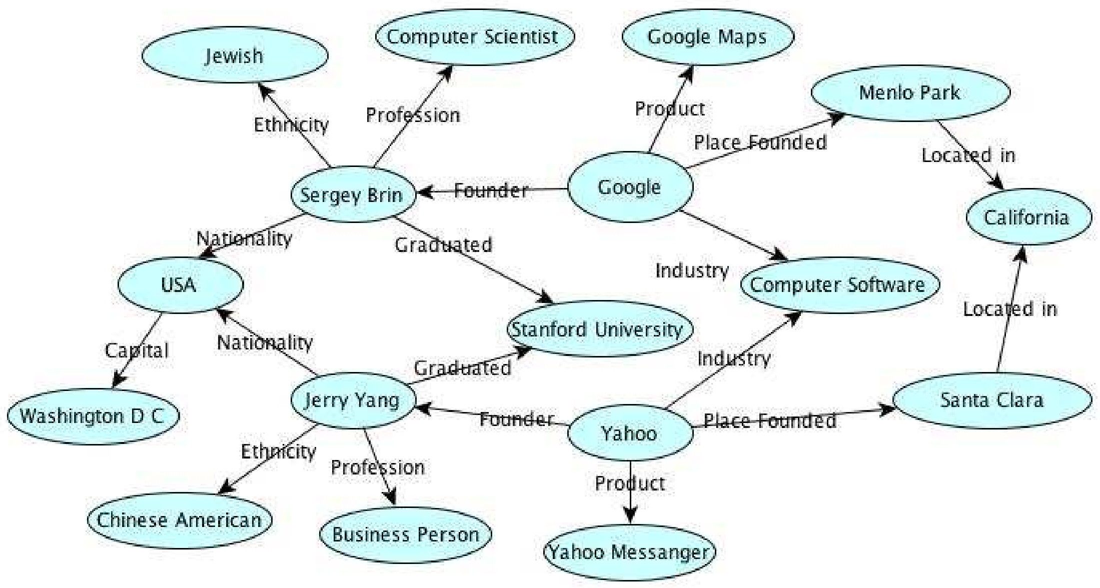


Figure 3.1 Entity-Relationship Graph

3.2 Neighborhood Graph

Given an entity-relationship graph G and a query tuple t, the corresponding neighborhood graph H_t is the weekly connected sub graph of G that consists of all undirected paths in G of length d or smaller, with at least one end point each such path being a query entity in t. The path length threshold d is an input parameter. Figure 3.2 shows neighborhood graph for query tuple {Jerry Yang, Yahoo} derived from entity-relationship graph G of figure 3.1 for d=2.

The neighborhood graph H_t in Figure 3.2 we have edge e1 ={Jerry Yang, USA} and label(e1) =*nationality*. One more edge e2 labeled *nationality* is also incident on "USA". The neighborhood graph on a complete real world dataset may contain many more such edges because a lot of people have "USA" as *nationality*. Among e1 and e2, e1 is an important edge because it is closer to input entity "Jerry Yang". The intuition is that edge e1 represents important relationship with respect to input entity "Jerry Yang" but e2 is deemed unimportant with respect to input entity. Similarly we can mark edge e3= {Sergey Brin, Stanford University}, label(e3)= *graduated* and edge e4= {Google, Computer Software}, label(e4)= *Industry* as unimportant edges. Figure 3.3 shows the reduced neighborhood graph after removal of unimportant edges from neighborhood graph in Figure 3.2.

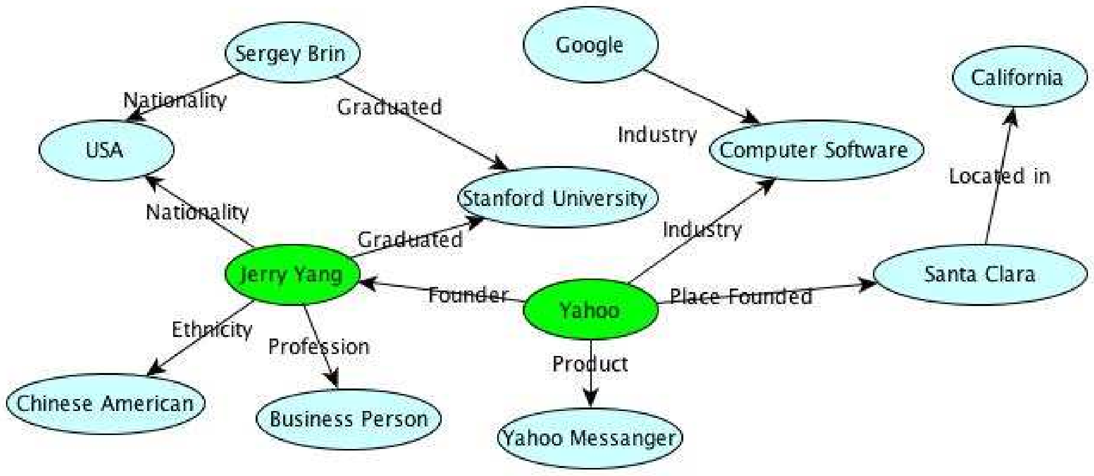


Figure 3.2 Neighborhood Graph of Depth d=2 for Input {Jerry Yang, Yahoo}

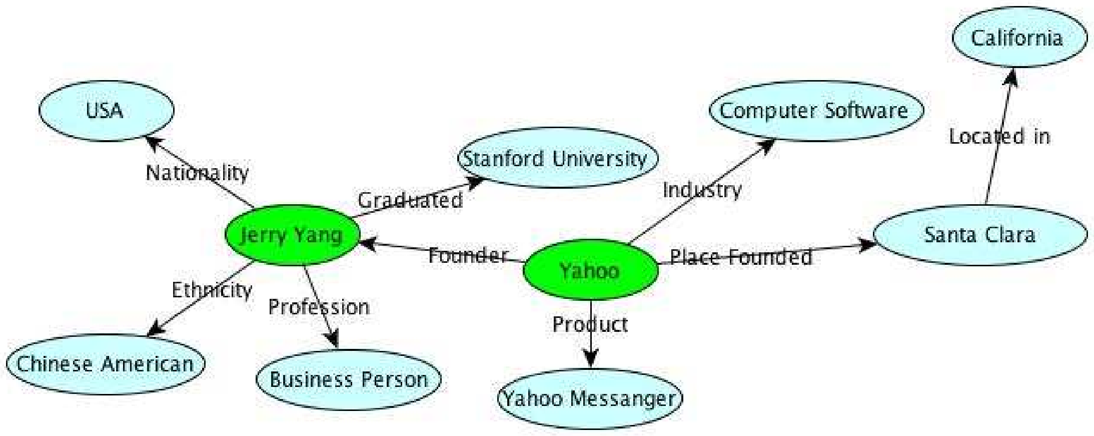


Figure 3.3 Reduced Neighborhood Graph from Figure 3.2

3.3 Maximal Query Graph

Even after removal of unimportant edges the reduced neighborhood graph H_t can be quite large even under a small path length threshold d. The query evaluation cost can be very high in finding approximate answers to such a large H_t. Hence we aim to drastically reduce the size of H_t and obtain a small subgraph that capture only important features of the input entities. This graph is termed as maximal query graph G_t.

To obtain maximal query graph, edges in the reduced neighborhood graph weighted by following measures.

Participation Degree- Participation degree p(e) of an edge e =(u, v) define as number of edges in data graph which share the same edge label and at least one of the two end nodes of edge e. Formally,

$$p(e) = |\{e' = (u', v') \mid label(e) = label(e'), u' = u \vee v' = v\}|$$

Inverse Edge Label Frequency- For edge e, it's defined as,

$$ief(e) = \log \frac{|E(G)|}{\#\, label(e)}$$

Where |E(G)| is the number of edges in data graph G and #label(e) is the number of edges in G with the same label as e.

So, given a input query tuple t, its reduced neighborhood graph H_t, and a parameter m, the maximal query graph G_t is a weekly connected subgraph of H_t that maximize total edge weight and it has m edges. Here weight of edge e define as,

$$w(e) = \frac{ief(e)}{p(e) \times d^2(e)}$$

Here d(e) is smallest distance of edge e from any input entity.

3.4 Query Graph

With respect to an entity-relationship graph G and its maximal query graph G_t for query tuple t a query graph Q is a weakly connected sub graph of G_t that contains all input entities in t. We denote set of all query graphs as Q_t. We can derive multiple query graphs corresponding to a query tuple. Figure 3.4 shows two query graphs.

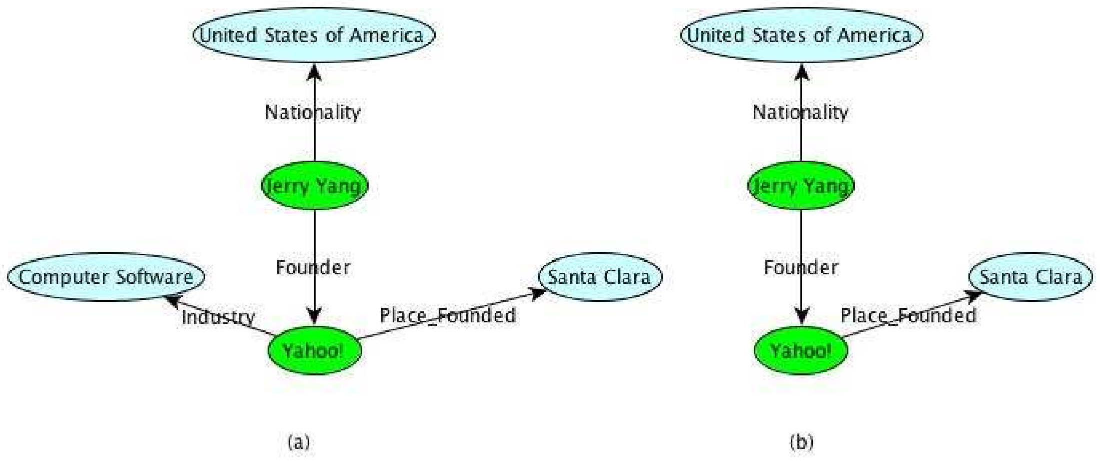


Figure 3.4 Examples of (a) Query Graph with 4 edges (b) Query Graph with 3 edges

3.5 Answer Graph

With respect to an entity-relationship graph G, its maximal query graph G_t for query tuple t and a query graph Q, an answer graph A is a weekly connected sub graph of G. A is also isomorphic to G except that entity label may not match. This definition of answer graph based on the intuition that an answer tuple is similar to input tuple, if they are in similar neighborhoods. The corresponding entities in two neighborhoods share common entity types or even the same entities, and they have the same relationship with each other. Figure 3.5 shows two answer graphs with respect to query graph in figure 3.4 (a) and Figure 3.6 shows two answer graphs with respect to query graph in figure 3.4 (b).

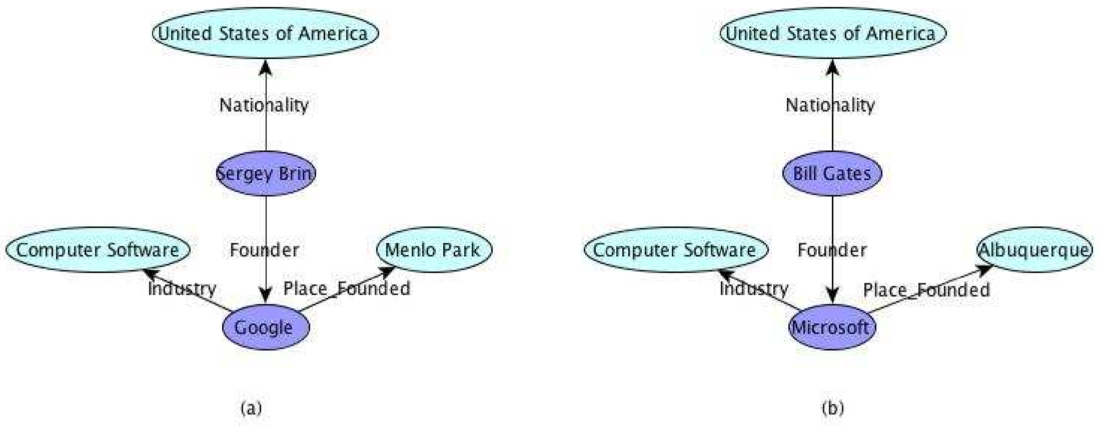


Figure 3.5 Answer Graphs for Query Graph in Fig 3.4(a): (a) Answer graph for answer tuple {Sergey Brin, Google} (b) Answer graph for answer tuple {Bill Gates, Microsoft}

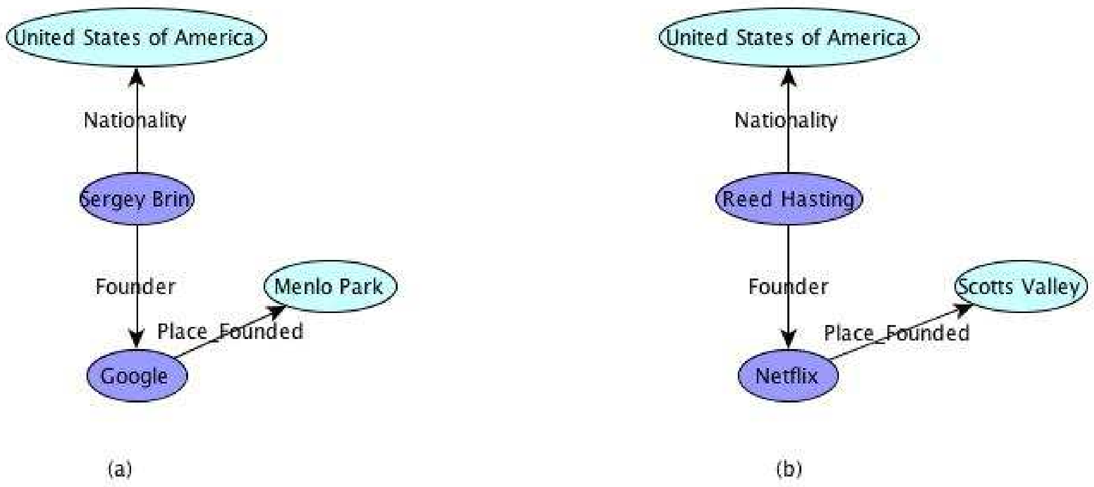


Figure 3.6 Answer Graphs for Query Graph in Fig 3.4(b): (a) Answer graph for answer tuple {Sergey Brin, Google} (b) Answer graph for answer tuple {Reed Hasting, Netflix}

3.6 Answer Tuple

Answer tuple is defined as bijection between a query graph and answer graph. So a query graph Q contains all input entities. Given an answer graph A, the entities corresponding to the input entities (based on the bijection f: V (Q) $\rightarrow$ V (A)) form the answer tuple. For example

{Sergey Brin, Google}, {Bill Gates, Microsoft}, {Reed Hasting, Netflix} are answer tuples from the answer graphs in figure 3.5 & 3.6.

3.7 Query Lattice

Given a maximal query graph G_t, the query lattice is a directed acyclic graph in which the root node is $G_{t,}$ and the other nodes are subgraphs of G_t. The leaf nodes of lattice are those query graphs that cannot be made any simpler and still keep all input entities connected. Figure 3.7(a) shows a maximal query graph, which contains two query entities in shaded circle and five edges A, B, C, D and E. Its corresponding query lattice is in Figure 3.7 (b). The root node of the lattice, denoted by ABCDE, represents maximal query graph itself. The two shaded nodes A and BC are leaf nodes of the lattice. The query lattice is constructed by first generating all leaf nodes, and then going all the way up to the root of the lattice. We define score of a node as sum of the edge weight. In subsequent two sections we will explain two different approaches for traversal of query lattice.

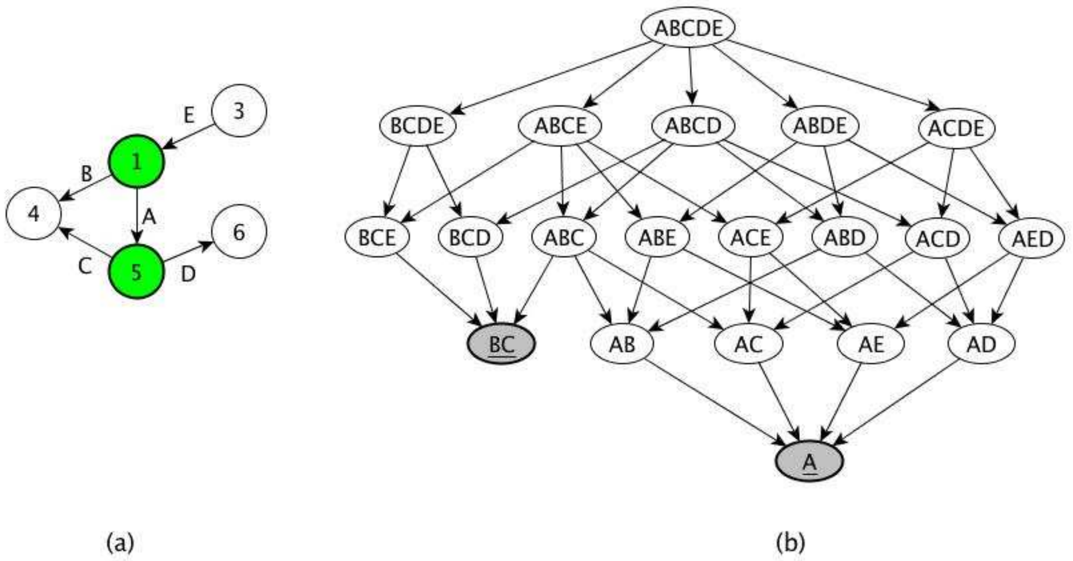


Figure 3.7 Example of (a) Maximal Query Graph (b) Query Lattice

3.8 Brute-Force Based Enumeration of Lattice Nodes

Here we employ breadth-first search (BFS) based bottom-up lattice exploration method as brute-force mechanism for evaluating the query lattice. We first start off with the leaf nodes and use the results of each lattice node to find the answer graphs of its parent. We complete the evaluation of every lattice node in a level before evaluating any lattice node in upper level. If traversing a lattice node does not yield any answer graph, all its ancestors are pruned out. Figure 3.7 shows example of query lattice. Here node ACD does not yield any answer graph, hence all its ancestors ABCD, ACDE and ABCDE won't be evaluated.

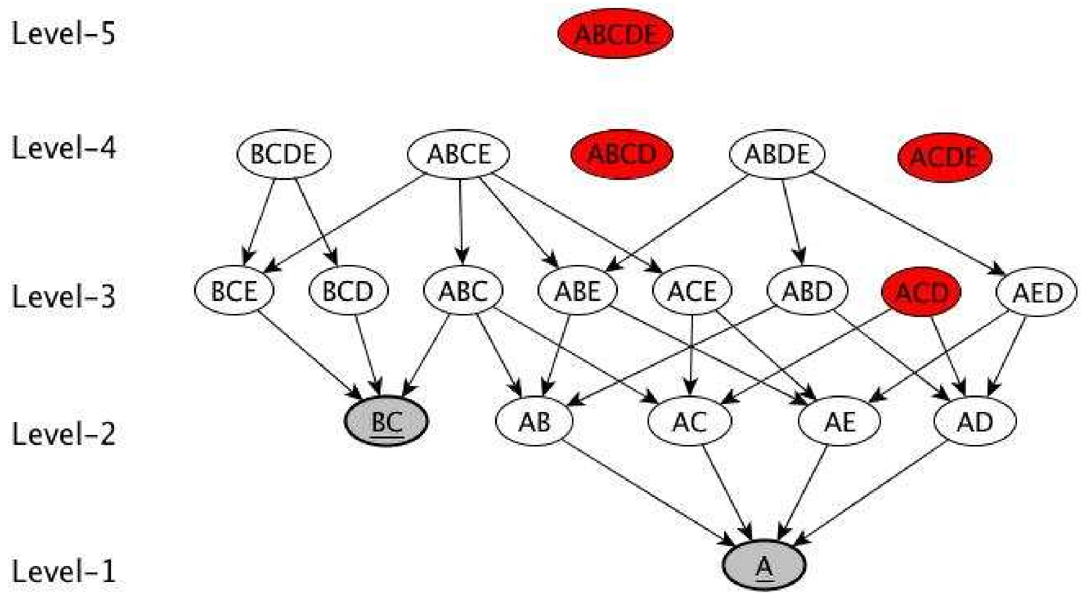


Figure 3.8 Brute-Force Based Query Lattice Enumeration

3.9 Best First Search Based Enumeration of Lattice Nodes

The Brute-force approach traverses many lattice nodes to find all the answer tuples, while we are only seeking top-K answer tuples. To traverse as few lattice nodes as possible and find only the top-K answer tuples, we follow a Best first search method. The Best-first search traversal always expands the most promising lattice node. A most promising node is a node that has the highest upper bound. Upper bound of a node is the score of its ancestor node that has the highest score among all its ancestors. Lower bound of a node is the score of the node itself. Upper boundary of a node in lattice is its ancestor node that has no ancestor in lattice. Clearly, at the beginning, upper boundary of each node is root of the lattice and their upper bound is the score of the maximal query graph. However, it changes as we traverse the lattice and detect null nodes. Figure 3.9 shows example of lower and upper bound values.

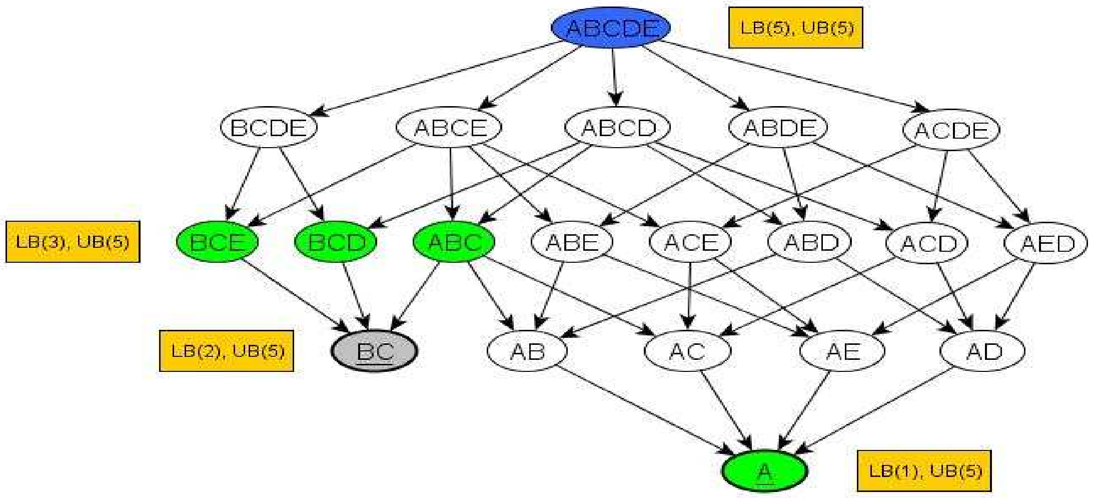


Figure 3.9 Initial Lower and Upper boundary in Lattice

Figure 3.10 (a) shows that node BCE is detected as a null node. So all ancestor nodes of BCE (BCDE, ABCE and ABCDE) removed from the lattice. Figure 3.10 (b) shows new upper boundary values.

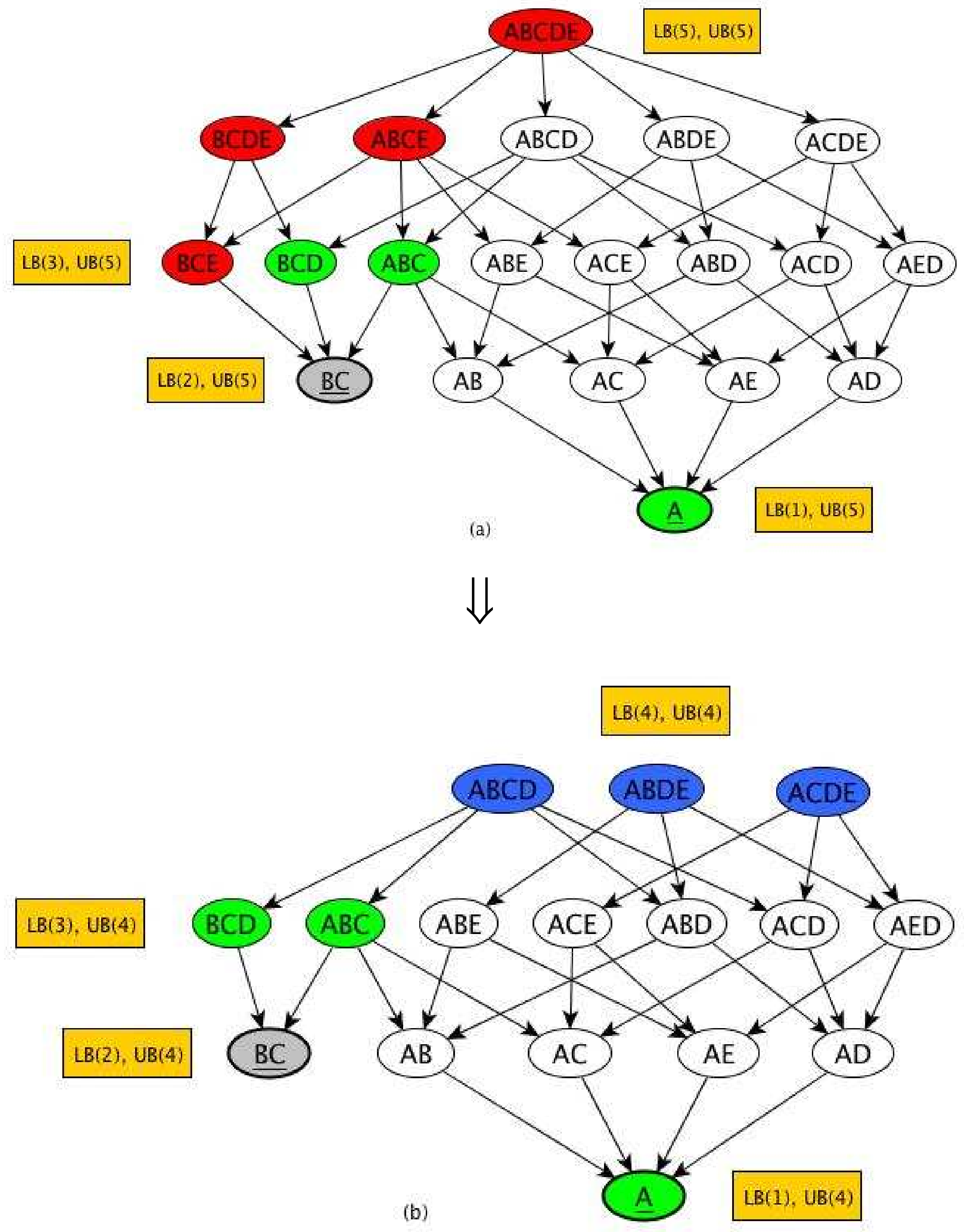


Figure 3.10 (a) Node BCE detected as NULL node (b) Recalculation of Upper Boundaries in Lattice

3.10 Top-K Answers Ranking

The score of an answer tuple t’ in answer graph A is the sum of two values – the total weight of edges in query graph Q of answer graph A and the extra credit given to matching nodes in A and Q.

$$Score(A) = \sum_{e \in E(Q)} W(e) + \sum_{\substack{e=(u,v)\in E(Q) \\ e'=(f(u),f(v))\in E(A)}} match(e,e')$$

Here match (e, e’) is for giving extra credit to matching nodes in Q and A defined as follows,

$$\text{match}(e,e') = \begin{cases} \frac{\mathbf{w}(e)}{|E(u)|} & \text{if } u=f(u) \\ \frac{\mathbf{w}(e)}{|E(v)|} & \text{if } v=f(v) \\ \frac{\mathbf{w}(e)}{min(|E(u)|,|E(v)|)} & \text{if } u=f(u), v=f(v) \\ 0 & \text{otherwise} \end{cases}$$

Node label matching increases overall size and complexity of the lattice. Hence algorithm first find top-K’ (here K’ is a heuristic based number larger than K) answers based on edge label match score. After that algorithm calculate node label match score on these top-K’ answers, re-rank them and return top-K answers.

CHAPTER 4

IMPLEMENTATION

In this chapter we are going to describe the implementation details of GQBE. We will also describe details of the Web-based demo developed for GQBE.

4.1 Freebase Data Dump Processing

We used Freebase dataset [7] of Sep, 2011. We cleaned it to keep only named entities (e.g., Stanford University} and abstract concepts {e.g., Jewish People}. Freebase also contains back edges between 2 entities, for example edge *inventor* connects node “Page Rank” to “Sergey Brin” and edge *inventions* connects “Sergey Brin” to “Page Rank”. We kept only one such edge on our dataset. Table 4.1 shows details of our dataset.

Table 4.1 Freebase Data Set Details

# Of Edges	# Of Nodes	# Of distinct Edges	# Of Node Type
46,708,421	28,483,132	5,428	3,307

4.2 Overview of GQBE Implementation

GQBE was implemented in Java. Figure 4.1 shows the process flow diagram for GQBE. The entire process can be explained as follows:

- System expects a query tuple as input.
- It first builds the neighborhood graph around the query tuple using a depth threshold d.
- It removes unimportant edges with respect to the query tuple from the neighborhood graph to create a reduced neighborhood graph.

- Maximal query graph is created from the reduced neighborhood graph that has maximum edge weight and at most m edges, where m is a heuristic based number.
- We have 2 different strategies for lattice evaluation. First one is breadth-first search based brute-force method and second is best first search method where only the most promising nodes in the lattice are evaluated. All answer graphs returned from this lattice evaluation are scored based on the edge label matching.
- After collecting answer graphs from lattice evaluation, node label matching is considered to score the answers, which are then re-ranked to find the final top-K answers.

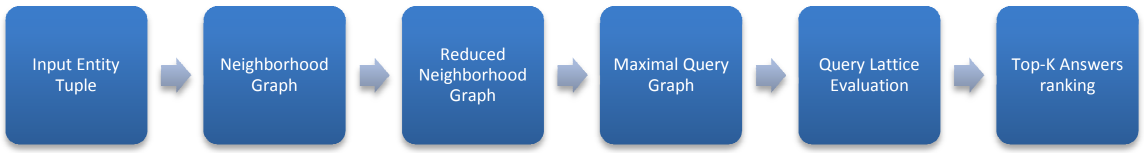


Figure 4.1 GQBE Process Flow Diagram

4.3 Web Demo for GQBE

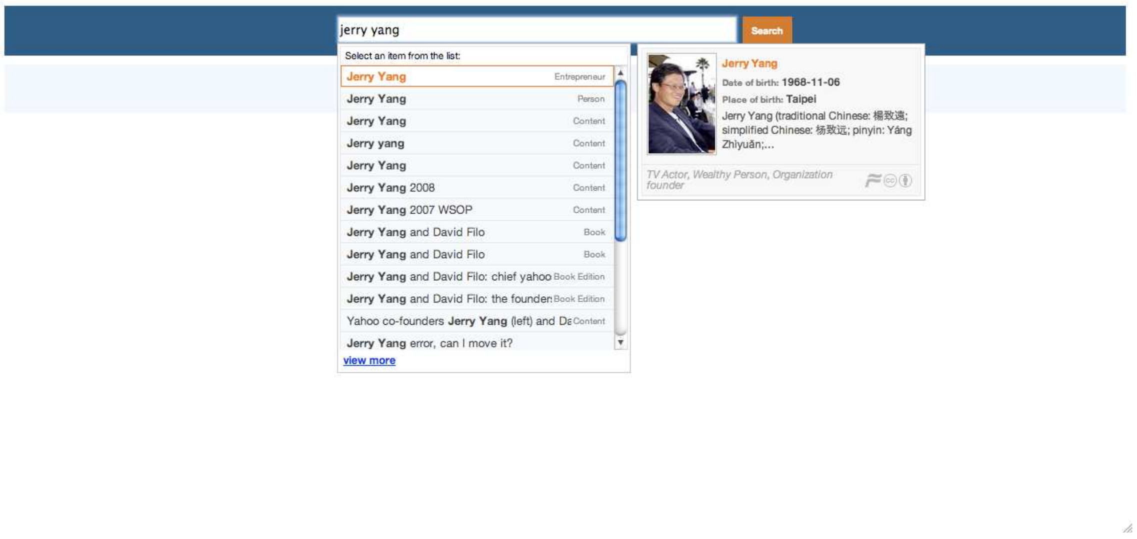
We have developed a web interface for GQBE. It has search box where a user can input example query tuples. The system fetches a list of ranked relevant answers. Users can see both edge label and node label based matching score for each answer. Users can also view the maximal query graph created by GQBE for the input query tuple and the answer graph of each answer tuple. Web demo was implemented using PHP and JQuery 1.7 JavaScript library [13]. We also use other external APIs whose details are presented in Table 4.2.

Table 4.2 GQBE API/Libraries Details

Component	Description
Freebase Suggestion API [9]	Gives search suggestion when a user types in the search box.

Table 4.2 – *Continued*

Freebase image API [10]	To show freebase image of answer entities.
Dracula Graph Library [12]	To show query and answer graphs in the browser.


jerry yang

Search

Select an item from the list:

Jerry Yang	Entrepreneur
Jerry Yang	Person
Jerry Yang	Content
Jerry yang	Content
Jerry Yang	Content
Jerry Yang 2008	Content
Jerry Yang 2007 WSOP	Content
Jerry Yang and David Filo	Book
Jerry Yang and David Filo	Book
Jerry Yang and David Filo: chief yahoo	Book Edition
Jerry Yang and David Filo: the founder	Book Edition
Yahoo co-founders Jerry Yang (left) and Da	Content
Jerry Yang error, can I move it?	

view more

Jerry Yang

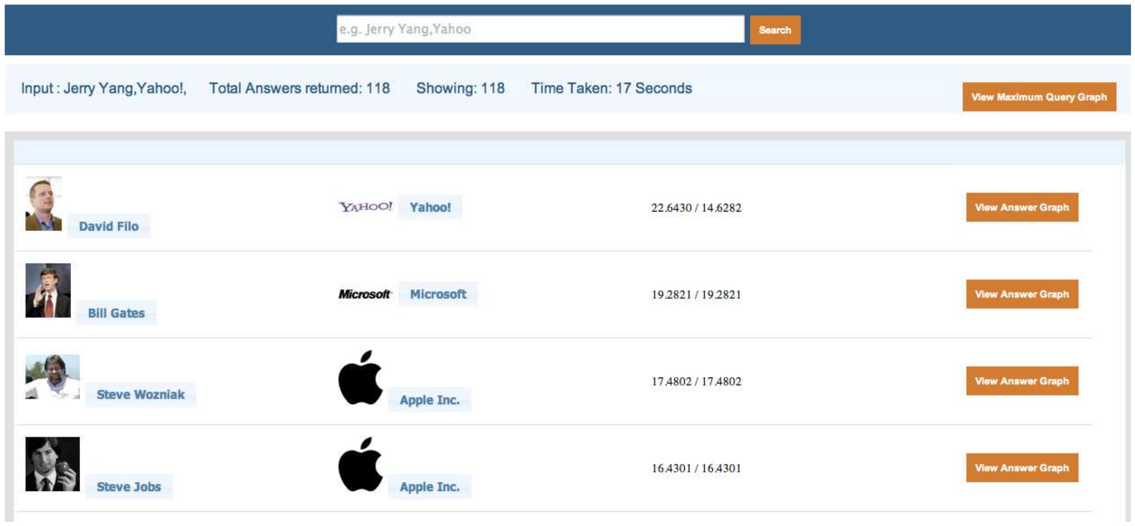
Date of birth: 1968-11-06

Place of birth: Taipei

Jerry Yang (traditional Chinese: 楊致遠; simplified Chinese: 杨致远; pinyin: Yáng Zhìyuǎn;...

TV Actor, Wealthy Person, Organization founder

Figure 4.2 Suggestions in Search Box


e.g. Jerry Yang,Yahoo

Search

Input : Jerry Yang,Yahoo!, Total Answers returned: 118 Showing: 118 Time Taken: 17 Seconds

View Maximum Query Graph

David Filo	Yahoo!	22.6430 / 14.6282	View Answer Graph
Bill Gates	Microsoft	19.2821 / 19.2821	View Answer Graph
Steve Wozniak	Apple Inc.	17.4802 / 17.4802	View Answer Graph
Steve Jobs	Apple Inc.	16.4301 / 16.4301	View Answer Graph

Figure 4.3 Top-4 Answers for Query {Jerry Yang, Yahoo}

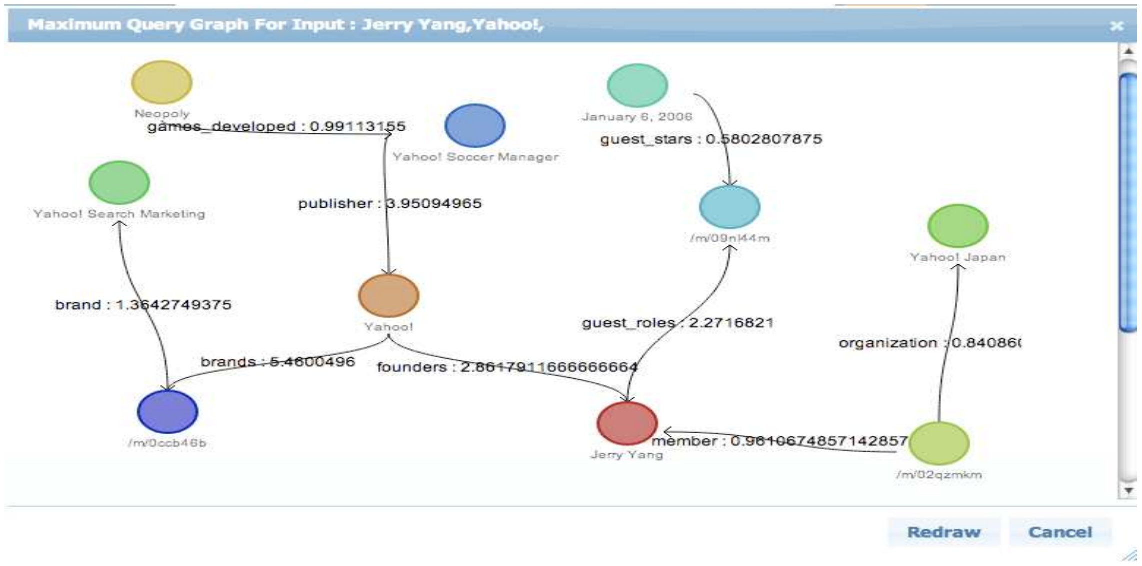


Figure 4.4 Maximal Query Graph for Query {Jerry Yang, Yahoo)

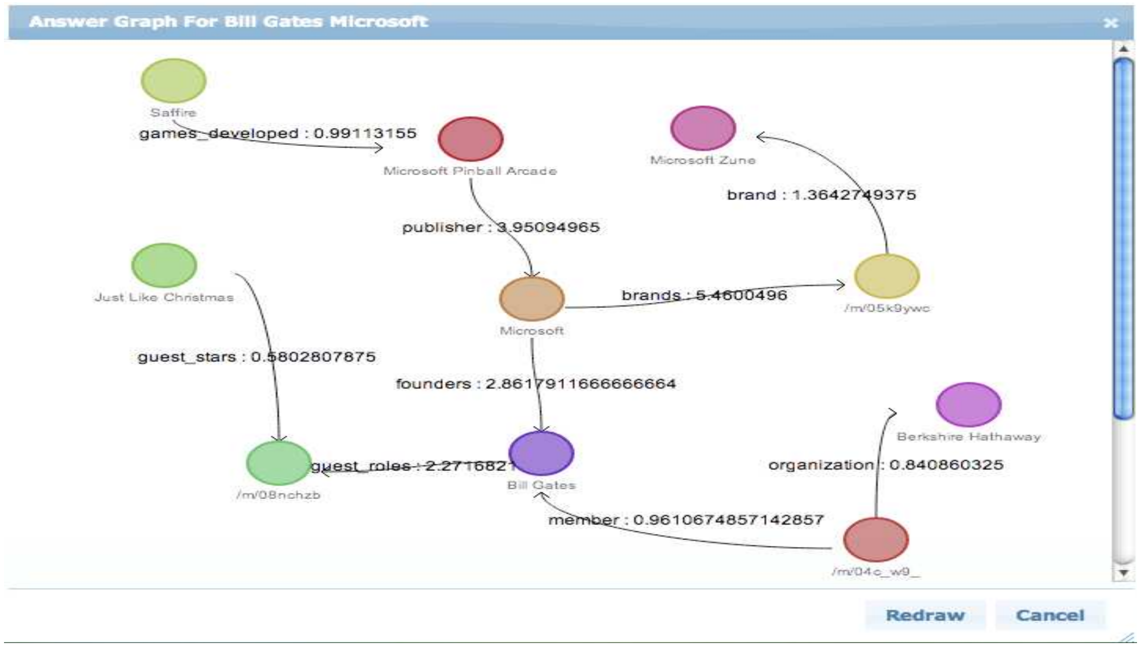


Figure 4.5 Answer Graph of Answer Tuple {Bill Gates, Microsoft}

CHAPTER 5

EVALUATION AND RESULTS

In this chapter I am going to describe details of evaluation metrics, query design, ground truth collection, Amazon Mechanical Turk user study. We will see comparison study between NESS and GQBE for precision and execution time.

5.1 Query Design and Ground Truth Collection

We used a set of 20 queries to evaluate the accuracy and efficiency of the system. These queries were designed based on various Wikipedia and Freebase tables. Each such table is a collection of tuples. In accuracy evaluation, for each table, we used one of its tuples as example tuples and its remaining tuples as ground truth .The 20 queries, their ground truth size and web URL's are summarized in table 5.1. Each query tuple consists of one, two, three entities. The queries cover diverse domain such as movies, people, companies, sports, awards, religions, universities, automobiles, and music.

Table 5.1 Queries and Ground Truth Details

Query	Ground Truth Size	Ground Truth URL
C	1240	http://www.freebase.com/view/computer/programming_language
TomKat	16	http://www.freebase.com/view/celebrities/supercouple
Jerry Yang, Yahoo!	8349	http://www.freebase.com/view/organization/organization_founder

Table 5.1 – *Continued*

Michael Phelps, Sportsman of the year	55	http://www.freebase.com/view/en/sportsman_of_the_year/-/award/award_category/winners
Gautama Buddha, Buddhism	621	http://www.freebase.com/view/religion/religion
Manchester United, Malcolm Glazer	40	http://en.wikipedia.org/wiki/List_of_English_football_club_owners
Boeing, Boeing C-22	89	http://www.freebase.com/view/en/boeing/-/aviation/aircraft_manufacturer/aircraft_models_made
David Beckham, A C Milan	94	http://www.freebase.com/view/en/ac_milan/-/soccer/football_team/current_roster
Beijing, 2008 Summer Olympics	41	http://www.freebase.com/view/olympics/views/olympic_host_city
Microsoft, Microsoft Office	200	http://www.freebase.com/view/en/microsoft/-/computer/software_developer/software
Jack Kirby, Iron Man	25	http://www.freebase.com/view/user/mahigupta/default_domain/views/comic_character_by_jack_kirby
Apple Inc, Sequoia Capital	300	http://www.freebase.com/view/user/mahigupta/default_domain/views/company_funded_by_sequoia_capital
Beethoven, Symphony no. 5	600	http://www.freebase.com/view/en/ludwig_van_beethoven/-/music/composer/compositions
Uranium, Uranium-238	26	http://www.freebase.com/view/user/mahigupta/default_domain/views/isotope_of_uranium

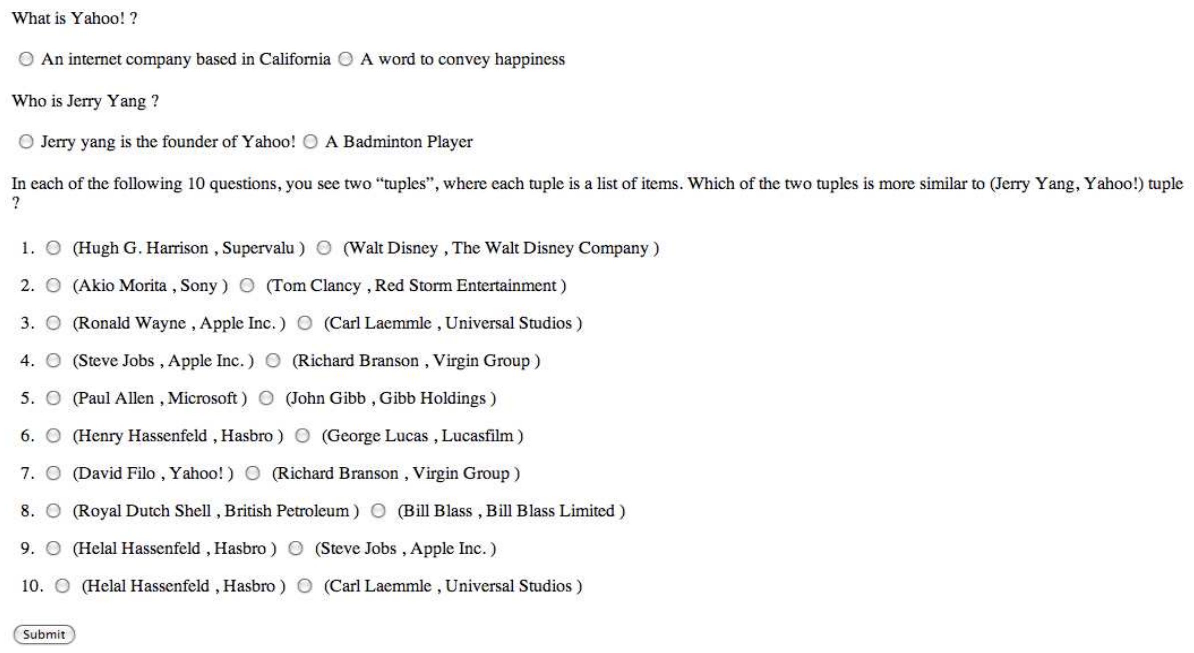
Table 5.1 – *Continued*

Microsoft Office, C++	300	http://www.freebase.com/view/user/mahigupta/default_domain/views/c_software
Dennis Ritchie, C	163	http://www.freebase.com/view/computer/programming_language_designer
Steven Spielberg, Minority report	40	http://www.freebase.com/view/user/mahigupta/default_domain/views/movie_directed_by_spielberg
Nike, Tiger Woods	20	http://www.freebase.com/view/user/mahigupta/default_domain/views/sponsored_by_nike
Donald Knuth, Stanford university, Turing Award	18	http://en.wikipedia.org/wiki/List_of_Turing_Award_laureates_by_university_affiliation
Ford motor, Lincoln, Lincoln MKS	25	http://www.freebase.com/view/base/ranker/rankerurlname/lincoln$002F2481808/-/automotive/make/model_s

5.2 User Study

We conducted an extensive user study using Amazon Mechanical Turk [26] to measure the accuracy of GQBE in the real world. We chose top-30 answers in each of the 20 queries and created 50 random pair per query. These 50 pairs broken down into 5 task of 10 questions each and 2,000 users were asked to rank the better answer in each pair with respect to the input query. In each form we had 1-2 screening questions about input entity tuple because we wanted each reviewer to be aware of input entity tuple. If reviewer answered screening question correctly, then only we accepted her review. Figure 5.1 shows example of one such task for query {Jerry Yang, Yahoo!}. Here first 2 questions are screening question about Jerry Yang and

Yahoo. It is followed by 10 questions, each have two random answer tuple. We measured accuracy of the ranking of answers by GQBE using Pearson Correlation Coefficient [17] between ranking and user feedback.



What is Yahoo! ?

○ An internet company based in California ○ A word to convey happiness

Who is Jerry Yang ?

○ Jerry yang is the founder of Yahoo! ○ A Badminton Player

In each of the following 10 questions, you see two "tuples", where each tuple is a list of items. Which of the two tuples is more similar to (Jerry Yang, Yahoo!) tuple ?

1. ○ (Hugh G. Harrison , Supervalu) ○ (Walt Disney , The Walt Disney Company)
2. ○ (Akio Morita , Sony) ○ (Tom Clancy , Red Storm Entertainment)
3. ○ (Ronald Wayne , Apple Inc.) ○ (Carl Laemmle , Universal Studios)
4. ○ (Steve Jobs , Apple Inc.) ○ (Richard Branson , Virgin Group)
5. ○ (Paul Allen , Microsoft) ○ (John Gibb , Gibb Holdings)
6. ○ (Henry Hassenfeld , Hasbro) ○ (George Lucas , Lucasfilm)
7. ○ (David Filo , Yahoo!) ○ (Richard Branson , Virgin Group)
8. ○ (Royal Dutch Shell , British Petroleum) ○ (Bill Blass , Bill Blass Limited)
9. ○ (Helal Hassenfeld , Hasbro) ○ (Steve Jobs , Apple Inc.)
10. ○ (Helal Hassenfeld , Hasbro) ○ (Carl Laemmle , Universal Studios)

Figure 5.1 Amazon Mechanical Turk User Study Form for Query {Jerry Yang, Yahoo)

5.3 Efficiency Metrics

We measured accuracy of the system by both comparing query results with ground truth and conducting user study. Accuracy was measured by four widely used metrics, including Precision-at-k (P@k) [14], Normalized Discounted Cumulative Gain (nDCG) [16], Mean Average Precision (MAP) [15], and Pearson Correlation Coefficient (PCC) [17]. We briefly review them below.

Precision-at-k: The ratio of top-k answers that belong to the ground truth list.

P@k = # answers in top-k present in Ground truth / k

Normalized Discounted Cumulative Gain (nDCG): It measures the accuracy of a ranked list by first computing the cumulative gain of relevant answer in the list. Relevance in our case is either 0 or 1 at each position. It penalizes the lower ranked relevant answers and aggregates the total gain. It is then normalized by the gain obtained for an ideal ranking which rank all ground truths at top.

Mean Average Precision (MAP): The average precision score for a query is based on P@k given by,

$$AveP(q) = \frac{\sum_{i=1}^{k} P@k(q,i)}{\#Ground - truths}$$

The MAP for a set of queries is the mean of AveP (q) over the queries given by,

$$MAP = \frac{\sum_{q} AveP(q)}{\#Queries}$$

Pearson Product-Moment Correlation (PCC): PCC we used in Amazon Mechanical Turk user study. For each query, we obtained top-30 answers by GQBE. We then generated 50 random pairs of these answers. We presented each random pair to 20 Amazon Mechanical Turk users and ask them to specify their preference in each pair. We then constructed two score lists for each query. One list X represents GQBE, computed by taking the difference between each pair's ranks in query result. The other list Y represents user's opinions, computed by taking the difference in numbers of users favoring the two answers in the pair. PCC is then computed between these two score lists given by,

$$r = \frac{\sum_{i=1}^{n}\left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{\sqrt{\sum_{i=1}^{n}\left(X_i - \bar{X}\right)^2\left(Y_i - \bar{Y}\right)^2}}$$

5.4 Setup

All the experiments were performed using dual-core 24 GB memory 2.0 GHz Xeon machine. The authors of NESS provided source code of NESS. To calculate edge weights, GQBE requires inverse edge frequency and participation of each edge in the data graph. We precomputed these values and stored them in files on secondary storage. GQBE loads these files into memory during computation.

5.5 Results and Comparison

Table 5.2 shows Precision comparison between GQBE and NESS for different values of K. NESS had almost equal precision for one-entity queries but its precision gets worse for two and three entities queries. Overall GQBE clearly beat NESS in all precision measures (P@k, nDCG, MAP) by big margin.

Table 5.2 Precision Comparisons Between GQBE and NESS

Top-K	GQBE			NESS		
	Precision-K	nDCG	MAP	Precision-K	nDCG	MAP
10	0.905	0.933	0.176	0.405	0.49	0.0723
15	0.87315	0.934	0.247	0.405	0.499	0.1
20	0.8368	0.9331	0.269	0.378	0.478	0.08
25	0.806	0.9357	0.299	0.3705	0.479	0.101
Over-All	0.85523	0.93395	0.24475	0.389625	0.4865	0.0832

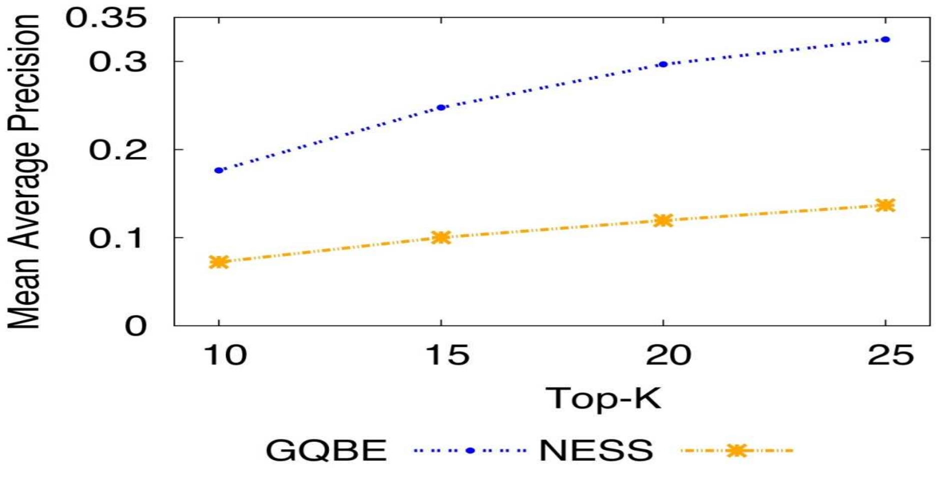


Figure 5.2 Mean Average Precision Comparisons Between NESS and GQBE

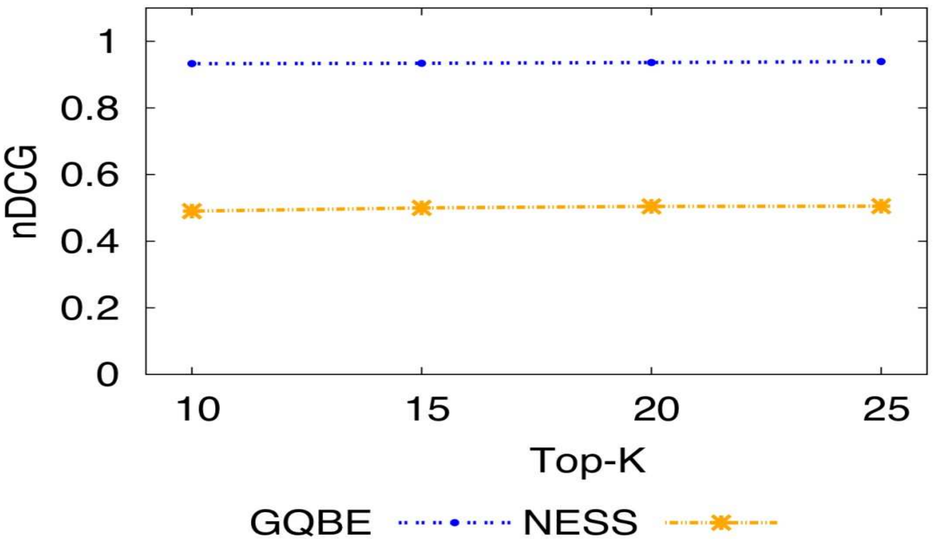


Figure 5.3 nDCG Comparisons Between NESS and GQBE

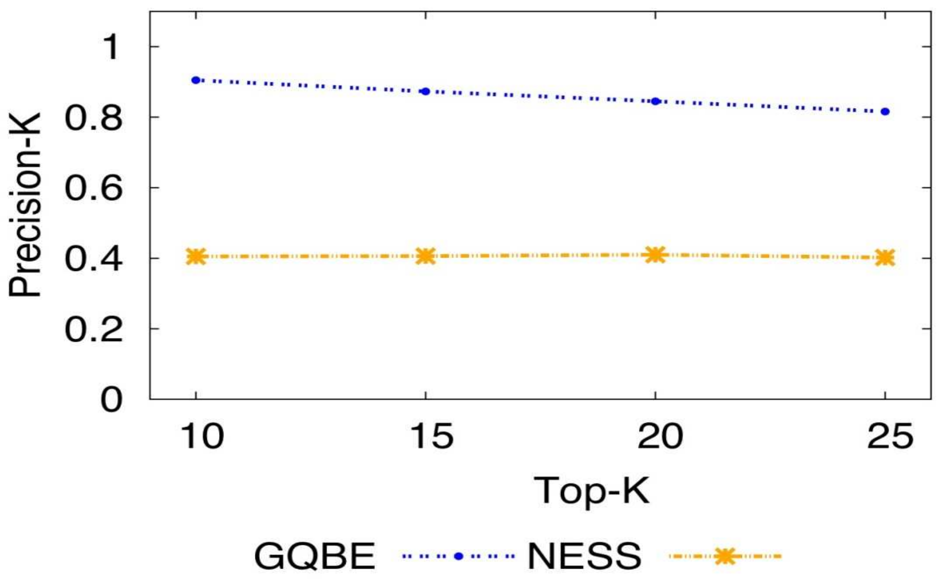


Figure 5.4 Precision-at-K Comparisons Between NESS and GQBE

The PCC value for each query and the entire system is summarized in Table 5.3. It can be observed that ranking of answer is better for 3 tuple queries, suggesting that more entities probably captures a better maximal query graph. It can also be observed that PCC value for {Apple Inc, Sequoia Capital} and {Beethoven, Symphony No. 5} is 0. This is because score of all top-30 answers were the same, indicating that all those answers were projected from same query graph. The overall PCC value for the ranking produced by GQBE across all the 20 queries is 0.497. Having a Pearson correlation coefficient over 0.5 is generally considered a strong positive ranking correlation [18]. A PCC of 0.497 indicates that GQBE produces a good ranking of answers, which is in line with what user expect in real world.

Table 5.3 Pearson's Correlation Coefficient (PCC) (Average=0.497)

Query	PCC	Query	PCC
Donald Knuth, Stanford University, Turing Award	0.79	Ford Motor, Lincoln, Lincoln MKS	0.78
Nike, Tiger Woods	0.6	Michael Phelps, Sportsman of the Year	0.8
Gautama Buddha, Buddhism	0.34	Manchester United, Malcolm Glazer	0.27
Boeing, Boeing C-22	0.06	David Beckham, A C Milan	0.26
Beijing, 2008 Summer Olympics	0.33	Microsoft, Microsoft Office	0.7710
Jack Kirby, Iron Man	0.578	Apple Inc, Sequoia Capital	0
Beethoven, Symphony No. 5	0	Uranium, Uranium-238	0.620
Microsoft Office, C++	0.43	Dennis Ritchie, C	0.29
Steven Spielberg, Minority Report	0.64	Jerry Yang, Yahoo	0.3
C	0.4	TomKat	0.65

Figure 5.5 shows total rum time comparison between NESS, GQBE and Brute force method. We study this with respect to size (number of edges) of maximal query graph. One can observe that running time of baseline suffered compared to GQBE and this is due to more number of lattice nodes baseline requires to evaluate. It is worth noting that the running time of GQBE and baseline do not increase consistently with increase in query graph size. This is because lattice evaluation time is not solely dependent on the number of edges, but it is also

dependent on the particular edges chosen in the maximal query graph. The running time of GQBE and NESS is comparable in most cases and NES performs a little better in some cases. This can be explained by the fact that NESS is an approximate graph querying system. It does not always try to find best score for an answer tuple with respect to a given maximal query graph. GQBE, on the other hand, always guaranteed to fetch the best score for an answer tuple.

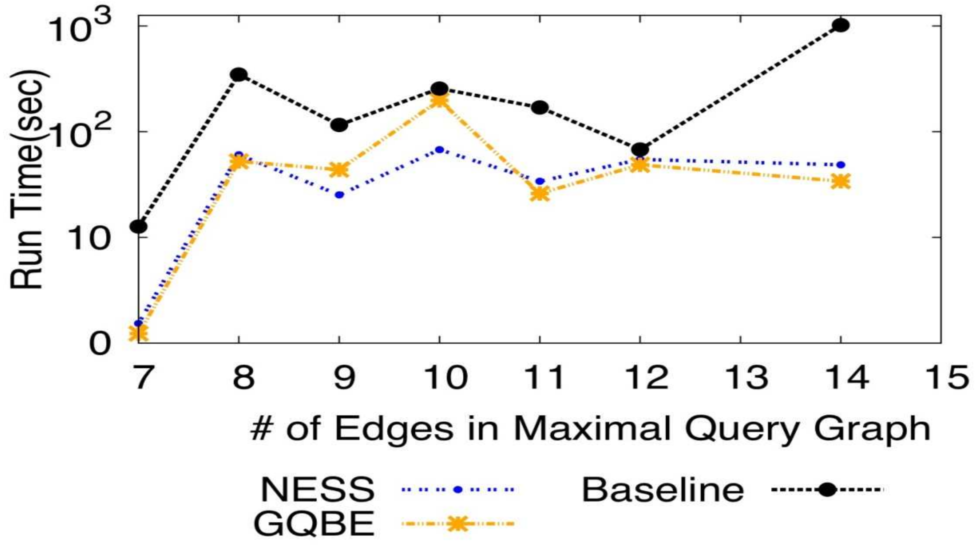


Figure 5.5 Run Time Comparisons Between NESS, GQBE and Brute-Force Method With Respect to Size of Maximal Query Graph.

Figure 5.6 shows time taken to discover maximal query graph as a function of number of entities in the query tuple, while also varying depth threshold d. Recall that maximal query graph generation involves first getting a neighborhood graph using d, and then using a greedy heuristic to obtain a much smaller subgraph. Once can observe that time increases when we have a higher d, since this increases the size of the neighborhood graph.

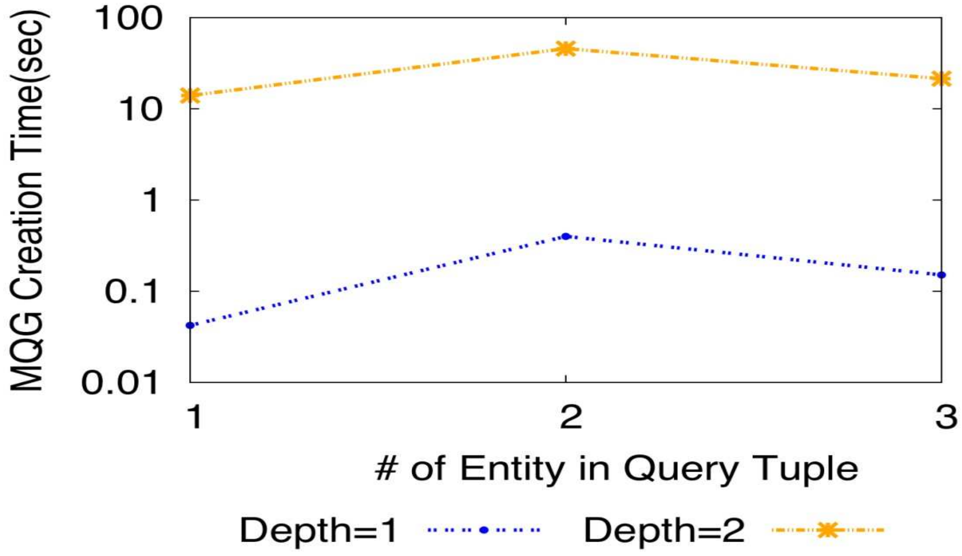


Figure 5.6 Time Taken to Create Maximal Query Graph With Respect to Number of Entities in Query Tuple.

Figure 5.7 shows comparisons between GQBE and Brute force method in term of number of lattice nodes evaluated. As we can observe, the number of lattice-nodes evaluated by GQBE is lesser than the brute-force. This is because in best-first strategy the choice of the next node to evaluate depend on the higher upper bound which force lattice traversal to greedily reach the top of the lattice. Evaluating fewer nodes has a direct impact on the running time of the algorithm, which is shown in the running time comparison of the two methods.

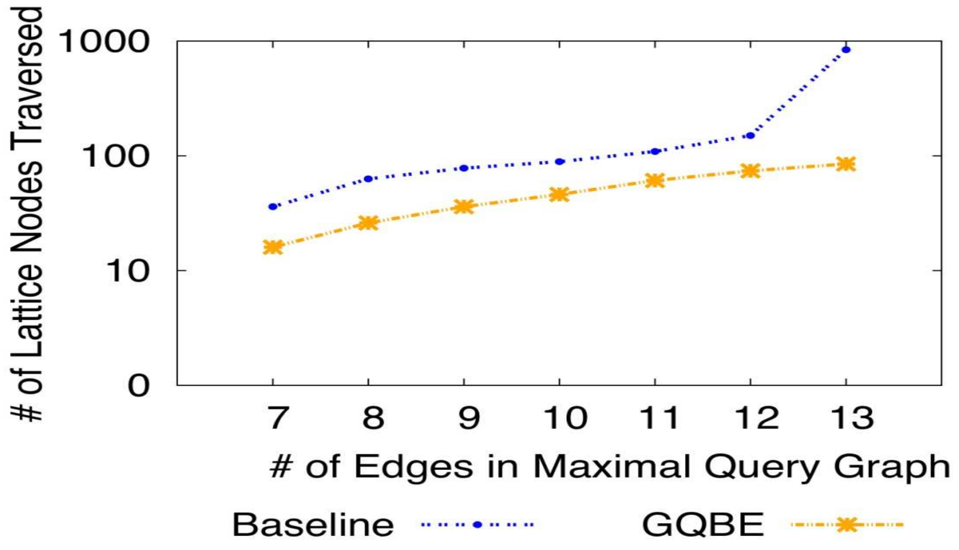


Figure 5.7 Comparisons of Brute-Force and GQBE With Respect to Number of Lattice Nodes Evaluated.

CHAPTER 6

CONCLUSION

GQBE is a system that queries entity-relationship graph by example entity tuples. GQBE automatically creates a maximal query graph based on the input example query tuple and finds the top-K matching answer tuples. Our experiments on large freebase dataset shows that GQBE clearly outperforms an adaption of a related system NESS in query answer accuracy. The runtimes of the two systems are comparable while the best first search algorithm outperforms the brute-force method by orders of magnitude.

As an initial step toward better usability of graph query system, GQBE saves users the burden of forming explicit query graphs, by allowing querying graphs by example entity tuple. As we see an unprecedented proliferation of entity data graphs in real world, GQBE will have profound impact on many future works.

REFERENCES

[1] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In SIGMOD, pages 901-912, 2011.

[2] Y. Tian and J. M. Patel. TALE: A tool for approximate large graph matching. In *ICDE*, pages 963–972, 2008.

[3] R. C. Wang and W.W. Cohen. Language-independent set expansion of named entities using the web. In *ICDM*, pages 342–350, 2007.

[4] M. M. Zloof. Query by example. In *AFIPS*, 1975.

[5] L. Chang, J. X. Yu, L. Qin, Y. Zhu, and H. Wang. Finding information nebula over large networks. In *CIKM*, 2011.

[6] http://www.freebase.com/

[7] http://download.freebase.com/datadumps/

[8] http://wiki.freebase.com/wiki/Main_Page

[9] http://wiki.freebase.com/wiki/Freebase_Suggest

[10] http://wiki.freebase.com/wiki/ApiImage

[11] http://wiki.freebase.com/wiki/ApiSearch

[12] http://www.graphdracula.net/

[13] http://jquery.com/

[14] http://en.wikipedia.org/wiki/IR_evaluation

[15] http://en.wikipedia.org/wiki/Information_retrieval#Mean_average_precision

[16] http://en.wikipedia.org/wiki/Discounted_cumulative_gain

[17] http://en.wikipedia.org/wiki/Pearson_correlation_coefficient

[18] http://en.wikipedia.org/wiki/Pearson_correlation_coefficient#Interpretation_of_the_size_of_a_correlation

[19] C. D. Manning, P. Raghavan, and H. Schtze. Introduction to Information Retrieval. Cambridge University Press, New York NY, USA 2008.

[20] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[21] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, pages 405–416, 2009.

[22] J. Pound, I. F. Ilyas, and G. E. Weddell. Expressive and flexible access to web-extracted data: a keyword-based structured query language. In *SIGMOD*, pages 423–434, 2010.

[23] A. Khan, Y. Wu, and X. Yan. Emerging graph queries in linked data. In *ICDE*, pages 1218–1221, 2012.

[24] E. Demidova, X. Zhou, and W. Nejdl. FreeQ: an interactive query interface for Freebase. In *WWW*, demo paper, 2012.

[25] L. Fang, A. D. Sarma, C. Yu, and P. Bohannon. REX: explaining relationships between entity pairs. In *PVLDB*, pages 241–252, 2011.

[26] https://www.mturk.com/mturk/welcome

[27] Y. He and D. Xin. SEISA: set expansion by iterative similarity aggregation. In *WWW*, pages 427–436, 2011.

[28] G. Kasneci, S. Elbassuoni, and G. Weikum. MING: mining informative entity relationship subgraphs. In *CIKM*, 2009.

[29] H. Tong and C. Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[30] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948, 2010.

[31] N. Jayaram, A. Khan, M. Gupta, C. Li, X. Yan, R. Elmasri. GQBE: Querying Entity-Relationship Graphs by Example Tuples. In Progress.

BIOGRAPHICAL INFORMATION

Mahesh Gupta completed his Bachelor of Technology in Information Technology from Maulana Azad National Institute of Technology (MANIT) Bhopal, India in May 2007.As a software engineer he joined Wipro Technologies, Bangalore India and worked 2 years 11 months as SQL developer. He started his Master in Computer Science at the University of Texas at Arlington in Fall 2010 and joined The Innovative Databases and Information System Research (IDIR) Lab at UT Arlington in Summer 2011. He also worked as a Web Developer Graduate Research Assistant in Center for online Development at UT Arlington from May 2011.His research involves Entity-Relationship graph, Web Search and Information Retrieval.