Error-tolerant Exemplar Queries on Knowledge Graphs

by

Zhaoyang Shao

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Edge-labeled graphs are widely used to describe relationships between entities in a database. We study a class of queries on edge-labeled graphs, referred to as exemplar queries, where each query gives an example of what the user is searching for. Given an exemplar query, we study the problem of efficiently searching for similar subgraphs in a large data graph, where the similarity is defined in terms of the well-known graph edit distance. We call these queries *error-tolerant exemplar queries* since matches are allowed despite small variations in the graph structure and the labels. The problem in its general case is computationally intractable but efficient solutions are reachable for labeled graphs under well-behaved distribution of the labels, commonly found in knowledge graphs. In this thesis, we propose two efficient exact algorithms, based on a filtering-and-verification framework, for finding subgraphs in a large data graph that are isomorphic to a query graph under some edit operations. Our filtering scheme, which uses the neighbourhood structure around a node and the presence or absence of paths, significantly reduces the number of candidates that are passed to the verification stage. We analyze the costs of our algorithms and the conditions under which one algorithm is expected to outperform the other. Our cost analysis identifies some of the variables that affect the cost, including the number and the selectivity of the edge labels in the query and the degree of nodes in the data graph, and characterizes the relationships. We empirically evaluate the effectiveness of our filtering schemes and queries, the efficiency of our algorithms and the reliability of our cost models on real datasets.

Acknowledgements

I would like to express my sincere gratitude to all the people who contributed to the work described in this thesis. First and foremost, I wish to express my sincere thanks to my supervisor, Dr. Davood Rafiei for his patience, motivation, and continuous support for my graduate studies. Besides my supervisor, I would like to thank the rest of my thesis committee: Dr. Lorna Stewart and Dr. Denilson Barbosa, for their interest in my work and insightful comments.

My sincere thanks also goes to Dr. Themis Palpanas and Dr. Davide Mottin. They have helped me with ideas and implementation, and I learned a lot through their research work.

Finally, I would like to thank my parents and friends for their selfless support and encouragement to finish my graduate studies.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Motivation. Graphs are widely used to model relationships, for example between chemical compounds and organisms, objects and scenes in images, entities in a knowledge graph, subjects and objects in Resource Description Framework (RDF)[19] or triplet stores, functions and subroutines in a piece of software, etc.

In particular, RDF is a graph data model for representing and exchanging information on the Web. The model intrinsically represents a labeled, directed multi-graph. The core structure of RDF is a set of triples, each consisting of a subject, a predicate and an object.

Knowledge Graphs, which maintain structured data about entities, are becoming a common support for browsing, searching and knowledge discovery activities. Search engines, such as Google, Yahoo! and Bing, use knowledge graphs to complement search results with facts about entities. Examples of real-world knowledge graphs include DBpedia[1], YAGO[10], Freebase[4] and Probase[37].

An important problem that arises in graph database models is finding graph structures that are similar to a query graph. Searching for similar rather than exact matches of a query is more desirable when data is noisy or inconsistencies are allowed. Here are some example scenarios.

1. In computational biology, annotating, indexing and searching for sub-

graphs in large networks generated with high throughput experiments is a challenging problem for biologists. To be specific, biologists want to identify well-characterized pathways or patterns in less studied model organisms[9]. In protein-protein interaction networks where vertices and edges represent proteins and iterations respectively, it is useful to query for pathways or patterns from well studied model organisms in other unfamiliar organisms with known protein-protein interaction networks. However, the data can be highly noisy because of possible errors in data collection and different thresholds used in experiments. Missing interactions are common and it is very difficult to clean the data. Despite the noise, searching for similar biological structures may enable a biologist to understand and identify pathways/patterns in not so well studied model organisms.

2. In molecular chemistry, a challenging problem is to identify similar molecular structures of a target molecule in a large molecular graph database. In molecular graph, the vertices and edges may represent atom and covalent bonds respectively. Identifying similar molecular structures of a molecule can help chemists understand and learn the similarities and differences between two different molecular structures and may enable a chemist to design new molecular structures[2].

3. Social networks represent a particular graph model where profiles and links between them are represented by nodes and edges respectively . Social networks play an increasing role in many areas of computer science applications. An important aspect of these applications relies on identifying similar nodes or structures. Searching for similar subgraph may help to identify communities and to predict the network dynamics[33].

4. In object-oriented programming, multiple objects of the same or different classes are handled by developers and testers. In the object dependency graph of a program, each object is a vertex and the interaction between two objects through a method call is an edge. This dependency

graph is helpful for developers and testers to understand the flow of the program and identify bugs. Querying for structure similar to a typical pattern may help developers and testers to quickly locate the bugs in programs[28].

Similarity Search. In all aforementioned scenarios, one needs to identify the subgraphs in a data graph that are similar to a query graph. A number of similarity measures have been proposed[11, 5, 29], of which the graph edit distance[12] is the most general and widely accepted similarity measure. Graph edit distance is defined as the number of operations (i.e. the deletion, insertion and substitution of nodes or edges) that needed to transform one graph into another. In nearest-neighbour classification, graphs may be classified into different categories based on their edit distances to graphs in different categories. Also, graph edit distance can be used to describe the pattern similarity in the context of kernel methods of machine learning. Graph edit distance has found applications in a large number of domains[26], such as support vector machines for classification and kernel principal component analysis for pattern analysis. A valuable feature of graph edit distance is its error tolerance, allowing user information needs to be captured in the presence of noise and distortion. This thesis uses the graph edit distance as its similarity measure between graphs. It has been proved that edit distance is suitable for error-tolerant graph matching in many applications[25, 31].

There is a large body of work on subgraph similarity search[34, 43, 22, 17, 18]. TALE[34], which indexes each node of a data graph with the node neighbourhood information (such as adjacent node labels, degrees, etc.), allows matching a subset of query nodes before progressively extending these matches. SAPPER[43] proposes a method to find approximate graph matches where the number of missing edges in the subgraphs is no more than a threshold. The authors take advantage of pre-generated random spanning trees to avoid a cost for searching overlapping graphs and a carefully designed graph enumeration order to minimize unnecessary searches. A set-cover based inexact subgraph matching technique is introduced in SIGMA[22]. The basic idea is to decompose each query to a set of edge features and look for collections of such feature sets in a data graph. This converts the subgraph search problem into the set cover problem. However, both SAPPER and SIGMA can only deal with missing edges for inexact graph matching and these techniques are not applicable in many scenarios. For example, consider searching for isomorphic matches of the query "Ruby influenced Swift" in the Freebase knowledge graph[4]. This search in TALE will return the query itself. The same query in SAPPER or SIGMA will return the nodes "Ruby" and "Swift", providing the information that users already know about. The Exemplar queries of Motin et al.[23] proposes a query paradigm, where a query is an example of the expected query answer. More formally, an exemplar query returns all isomorphic matches where the matching edge of a query has the same label as the query. For instance, the exemplar query "Ruby influenced Swift" (as shown in Figure 1.1) returns all matches of the form "A influenced B", e.g. "D influenced Swift", "Java influenced Closure", etc. However, a perfect matching of the labels of all query edges can be too strong constraint and may not retrieve many desired matches.

Error-Tolerant Exemplar Queries. In this thesis, we propose a method that overcomes the problems mentioned above, through the use of graph edit distance operations. Introducing edit operations in exemplar queries may significantly expand the search space: on a data graph with L distinct labels, a naive solution is to run an exemplar query for every edit, i.e. $O(L^t\binom{|E_q|}{t})$ exemplar queries for a query with $|E_q|$ edges and edit distance threshold t, which would be prohibitively expensive. Therefore, novel techniques are necessary in order to provide efficient and scalable solutions. Since edges with mismatching labels can match when edit distance operations are allowed, we call our queries *error-tolerant exemplar queries (ETEQ)*. A detailed motivating example is given in Section 1.2.

Given that ETEQ generalizes exemplar queries, the queries in ETEQ are applicable in many domains where one does not have a clear idea of what is being searched, but has a starting element in the result set. For example, the query "Ruby and D influenced Swift" with ETEQ can give the influence relationships between programming languages as well as other relevant relationships that may be retrieved when edit operations are allowed. Also, ETEQ can help existing search engine services improve in two ways. First, search engines can append the results of ETEQ to their results, which can increase the recall and may better capture the users' information needs. Second, the results of ETEQ can be considered related, or additional queries that are suggested by the search engines. For instance, when a user searches information about "relationship between Ruby and Swift", current search engines will return the results that mention the relationship. ETEQ can provide relationships of other programming languages, e.g. Java and Closure, Swift and Ruby, etc.

We identify a few challenges with efficiently evaluating ETEQ. First, the number of joins for a query with $|E_q|$ edges is in the order of $O(|E_q|)$. This becomes a computationally intensive process for large values of $|E_q|$. Second, allowing edit operations further increases the size of the search space, as well as the space overhead for the intermediate results. We address these challenges by (1) constructing efficient indexes and sketches for filtering candidates; and (2) developing accurate estimates for query selectivity and cost. Accordingly, we develop two algorithms for efficiently evaluating ETEQ: these algorithms explore the overlap among query transformations under different edit operations, and can effectively reduce the search space and minimize the overall cost.

Our contributions can be summarized as follows:

1. We extend exemplar queries with edit distance operations to support error-tolerant searches on graph data.

2. We propose two efficient algorithms for ETEQ based on a filtering-and-verification framework, and study efficient pruning strategies that use the neighbourhood structure and the paths to filter unqualified results.

3. We develop cost models that allow us to compare the cost of our algorithms and across different queries without actually running the algorithms.

4. We analyze our algorithms using our cost model and study the conditions under which one algorithm is expected to outperform the other.

5. We perform a thorough experimental evaluation of the effectiveness of our filtering schemes, the performance and the scalability of our algorithms and the reliability of our cost model.

1.2 Motivating Example

Consider a search scenario where one wants to find more information about programming languages. The user, if not familiar with the area, may try “programming languages’ basic information”. But this query will most likely return documents discussing programming languages in general terms. The user may instead provide an example result. Thus, she can formulate a query with all basic information about Swift as shown in Figure 1.1. This query, typed into a search engine, will return results about Swift (or maybe Ruby, Scala, etc.), but no results covering other languages.

Using exemplar queries allows us to find relevant answers matching all query edges. However, the given relationships in the query only holds for Swift. In other words, there is no other relevant answer that perfectly matches all query edges and their labels. Consider the candidate answer shown in Figure 1.1 regarding the Closure programming language. Although the relationship between Closure and Rich Hickey (“developer”) does not appear in the query graph and there are only two edges labeled ”influenced” in the Closure figure (three edges labeled ”influenced” in the Swift figure), the candidate has very similar structure to the query and is very likely an answer that users will find relevant. Exemplar queries cannot find such relevant answers. Also, it might be difficult for the users to describe the query with accurate relationships between entities, e.g. relationship between Swift and Treehouse.

Thus, there is a need to devise a method for searching relevant answers the user is interested in from a single example that may contain errors and mismatches in labels.

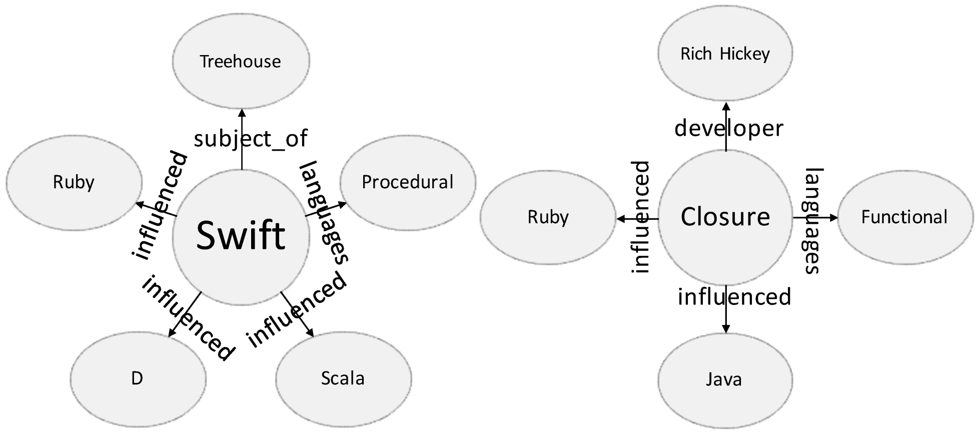


Figure 1.1: ETEQ and relevant answer.

1.3 Definitions and Problem Statement

Data sources that are in the form of entities and relationships may be encoded as labeled directed graphs using nodes to represent entities and edges to represent the relationships.

Definition 1. *(Knowledge Graph) A knowledge graph $G = \langle V, E, L \rangle$ is a directed labeled graph, where V denotes a set of nodes, $E \subseteq V^2$ is a set of edges, and L is a labeling function that maps each node in V and each edge in E to a label.*

Unless explicitly stated otherwise, the terms *graph*, *knowledge graph* and *edge-labeled graph* are used interchangeably in this thesis. Also, the term query in this thesis denotes a directed labeled graph. An example of query and data graph is shown in Figure 3.1.

Definition 2. *(Path) A path in a graph is a (finite or infinite) sequence of edges which connect a sequence of vertices which are all distinct from one another.*

An example of a path in a graph is $(q_1 \rightarrow q_2 \rightarrow q_3)$ as shown in Figure 3.1.

Definition 3. *(Edge-preserving Isomorphism) A graph $G_1 = \langle V_1, E_1, L_1 \rangle$ is edge-preserving isomorphic to a graph $G_2 = \langle V_2, E_2, L_2 \rangle$, denoted as $G_1 \simeq G_2$,* *if there is a bijective function $f\colon V_1 \to V_2$ such that $u \xrightarrow{l} v \in E_1$ if and only if $f(u) \xrightarrow{l} f(v) \in E_2$.*

Unless explicitly stated otherwise, the terms *isomorphism* and *edge-preserving isomorphism* are used interchangeably in this thesis.

Definition 4. *(Edge-preserving Edit Distance) The edit distance between two non-isomorphic graphs G and G' is the minimum number of edit operations that makes $G \simeq G'$, where edit operations include:*

1. *vertex insertion (to introduce a single new labeled vertex to a graph).*
2. *vertex deletion (to remove a single vertex from a graph).*
3. *vertex substitution (to change the label of a given vertex).*
4. *edge insertion (to introduce a new labeled edge between a pair of vertices).*
5. *edge deletion (to remove a single edge between a pair of vertices).*
6. *edge substitution (to change the label of a given edge).*

As an example, the edge-preserving edit distance between two graphs in Figure 1.1 is 2, because the label of "subject_of" in the query is deleted in the answer graph and one label of "influenced" in the query is substituted by the label "developer" in the answer graph.

Without loss of generality, we may limit the edit operations only to the queries. On the other hand, not all edit operations are applicable to exemplar queries. In particular, inserting an edge or a vertex to the query graph is not a meaningful operation under subgraph isomorphism; if there is no subgraph in the data graph that matches a query under an edit distance threshold, adding an edge to the query will not decrease the edit distance between the query and subgraphs in the data graph and thus is not going to change the result. Also the label substitutions are limited to the edge labels since the node labels are ignored in exemplar queries (this is because we are interested in answers that have the same structure as the query, but do not necessarily involve the same nodes[23]). This reduces the edit operations to edge deletion and edge label substitution. In general, each edit operation may have a different cost. For example, substituting a label may be less costly when the two labels are synonyms. That said, for the sake of simplicity of our presentation, we assume all edit operations have the same cost, and may sometime refer to the edit threshold t as the number of edit operations that are allowed.

Definition 5. *(Error-tolerant Exemplar Query) An error-tolerant exemplar query is a pair (Q, t) where Q is a graph and $t \in R$ is a threshold. The answer to query (Q, t) on a database D is the set of all subgraphs D_{sub} in D such that D_{sub} becomes edge-preserving isomorphic to Q after applying some edit operations to Q, D_{sub} or both, and the cost of those operations does not exceed the threshold t.*

An example of error-tolerant exemplar query is shown in Figure 1.1 with edit distance threshold set to 2.

In the rest of this thesis, we will refer to error-tolerant exemplar queries simply as queries. By query cost model, we mean a parametric equation that estimates the cost of an algorithm or a query plan, in terms of the number of operations (I/O and CPU) that is needed to evaluate the query.

Problem Statement: We aim to address the following two problems. (1) Given an ETEQ in the form of a query graph Q and an edit distance threshold t, we aim to efficiently retrieve all relevant answers in a data graph that are edge-preserving isomorphic to Q with at most t edit operations. (2) Given two algorithms for the problem in (1), we aim to compare their costs in terms of a cost model and find out the conditions under which one algorithm outperforms the other.

1.4 Overview

In this thesis, we extend exemplar queries with edit distance operations to support error-tolerant searches on graph data. To perform efficient error-tolerant searches on graph data, we propose two efficient algorithms based on a filtering-and-verification framework (exemplar queries with edit distance algorithm and wildcard queries with edit distance algorithm), and study efficient pruning strategies that use the neighbourhood structure and the paths to filter unqualified results. The empirical experiments show that: (1) wildcard queries with edit distance algorithm using both pruning strategies have the best performance with different experiment parameter settings; (2) both algorithms outperform SAPPER, which is a state-of-the-art algorithm for the problem of subgraph searches with edit distance constraints; (3) error-tolerant exemplar queries algorithms show its effectiveness comparing to exemplar queries for queries with introduced errors. Also, two types of cost models (exact cost model and upper-bounded cost model) are developed to compare the cost of our algorithms and across different queries without actually running the algorithms. We evaluate our cost models in terms of the correlation between our estimates and the actual costs. The empirical results show that: (1) our exact cost model is reliable for comparison of our algorithms when the number of query edges $|E_q| \leq 6$; (2) our upper-bounded cost model is reliable for the comparison of query costs when the number of query edges $|E_q| \leq 8$.

Chapter 2

Related Work

Graph-based data models have received much attention lately, and this has inspired more recent work on efficiently searching and querying graphs. Related works of this thesis, in particular, include subgraph isomorphism algorithms, graph edit distance, graph similarity search and querying knowledge graph.

2.1 Subgraph Isomorphism

Subgraph isomorphism problem is known to be NP-complete. Ullmann[35] proposed a backtracking-based algorithm using state space search method. However, the running time of this algorithm is, in general, exponential and is difficult to be applied for large data graph. VF2[8] is proposed to improve Ullmann's refinement by reducing the number of backtracks with the help of a forward checking technique. In these algorithms, they directly search for isomorphic subgraphs without preprocessing the data graphs.

There are many graph matching and searching algorithms proposed using indexing structures[7, 16, 32, 39, 41, 13], which fall into two categories: graph indexing and subgraph indexing. Given a query graph, graph indexing (e.g. TreePi[41], FG-index[7] and gIndex[39]) searches all data graphs to find graphs that contain or are contained by the query; while subgraph indexing (e.g. GraphGrep[13], TALE[34], GADDI[42]) indexes a large data graph so that similar subgraphs for given query graph can be efficiently retrieved. Our indexing techniques in this thesis that indexes neighbourhood and path information for large data graph is a type of subgraph indexing.

2.2 Subgraph Matching with Edit Distance Constraints

Subgraph isomorphism is a special case of graph edit distance with the edit distance threshold set to zero. Graph edit distance problem is known as NP-hard[40]. To compute the graph edit distance of two graphs, most of the existing algorithms use the best first search paradigm A*[14]. A* is originally used to find the minimum cost path between two nodes using heuristics. Since edit distance of two graphs is the minimum number of operations that one graph has to make to transform into another graph, it is intuitive to apply A* to find the edit distance of two graphs. A* explores the node mappings space like traversing an ordered tree. At each search stage, A* selects the best node mapping to expand, where the mapping induces the minimum edit cost. This step continues until every node in a graph has its mapping. In the simplest scenario of A*, the estimated future cost of choosing a node mapping is set to zero. In the other extreme, the future cost of choosing a node mapping is computed in exponential time, which is also unreasonable. To solve this problem, Riesen et. al.[30] proposed a new heuristic function to estimate the future cost using bipartite graph matching. The idea is to convert the selection of node mappings to assignment problem[20] and use Munkres' algorithm[24] to find the minimum cost of node mappings, which costs polynomial time in the worst case. Since the graph structure is not considered while choosing the node mappings, the estimated cost using bipartite graph matching is lower than the actual edit cost, this algorithm guarantees that the optimal edit cost can be found. However, as stated in the paper[27], these kind of algorithms are practical for computing the edit distance of two graphs with at most 12 vertices.

Most other existing methods adopt a filter-and-verification framework. Wang et. al.[36] proposed an efficient index for sparse data graphs. They decompose graphs to small κ-Adjacent tree patterns and use these κ-AT patterns to estimate a lower bound of their edit distance for candidate filtering. Zeng et. al.[40] proposed another method to compute the edit distance by transforming a graph to a multi-set of star structures, which is exactly a 1-gram defined by κ-AT; however, they apply bipartite matching instead of count filtering to derive the lower and upper bounds of edit distance for filtering and verification. Zheng et. al.[40] proposed a path-based q-gram index to derive a lower bound for candidates filtering. However, in most of these algorithms, they are targeting at data graphs that are small ($<$ 10K nodes) or sparse, the derived lower bound are not suitable for subgraph search and thus are not suitable for discovering information from RDF data graphs such as freebase.

The authors of SAPPER[43] proposed a method to find the approximate graph matches, i.e. the number of missing edges in the subgraphs is no more than some threshold. They take advantage of pre-generated random spanning trees to avoid the cost for searching overlapped graphs and a carefully designed graph enumeration order is designed to minimize unnecessary searches. Mongiovi et. al.[22] introduced a set-cover based inexact subgraph matching technique, called SIGMA. The basic idea is to decompose the query to a set of edge features and look for collections of such feature set in the given graph. This converts graph search problem into the set cover problem. However, both SAPPER and SIGMA can only deal missing edges for inexact graph matching.

2.3 Similarity-based Search

Similarity based search in large graphs has been studied in the past under various settings. TALE[34] introduces a neighbourhood based index (NH-Index) where it matches important vertices of a query graph first before extending the match progressively. In our approach, the importance of each node is considered as same and thus this technique is not applicable in our case.

In NESS[17] and Nema[18], the match of a query graph may not necessarily be isomorphic to the query graph in terms of label and topologically equality. NESS[17] introduces a novel neighbourhood-based similarity measure by vectorizing nodes according to the label distribution of their neighbours and extends the similarity notion to graph by finding the embeddings in the target graph that maximize the sum of node matches. Similarly, Khan et. al.[18]

introduces a similarity measure that preserves the proximity of node pairs and label information using k-hop neighbourhood of each node. In both above methods, a node is considered as a match of another node when the labels in their neighbourhood match regardless of the node structure in their neighbourhood. The difference between two measures is that in NESS, a mismatch in the neighbourhoods of two nodes costs the same irrespective of the neighbourhood while the cost of a mismatch in NeMa depends on the distance to the nodes. In our approach, the label distribution and node structure are both considered when matching two nodes. Our approach is related to NESS in which the query node's neighbourhood information is used to prune graph candidate nodes that are not related to those in the query. Our experiments confirm that combining both indexes performs better than either index alone.

Our work is also related to exemplar queries in the way that we both find relevant answers with similarity measure based on edge label. Exemplar queries can only find relevant answers that are edge-isomorphic to the query, while our algorithms in this thesis can find relevant subgraphs that are edge-preserving isomorphic to the query after some edit operations.

Jayaram et al.[15] present an algorithm that takes a set of entities (instead of a graph) and finds the best matching subgraph that includes those entities. Yahya et al.[38] introduce a method for querying and ranking on extended knowledge graph that combine knowledge graph and textual web contents. The resulting subgraph of both above two methods may be used as an exemplar query. This work is orthogonal to ours and may be combined with ETEQ, for more efficiently developing queries.

Chapter 3

Exemplar Queries with Edit Distance Constraint

3.1 The Basic EXED Algorithm

Given a data graph $G = (V, E)$, a query Q and the edit distance threshold t, a naive approach to find subgraphs that are within t edit distance of the query Q is to compare the query with every subgraph in the data graph G. Our basic exemplar queries with edit distance constraint algorithm (EXED) randomly chooses one node n_q from the query as a seed, instead of comparing the query with an exponential number of subgraphs in the data graph. Subsequently, it considers all nodes of the data graph one by one as possible mappings of the node n_q. For each such node n_g in V, it checks if there exists a subgraph that contains n_g and is isomorphic to the query with at most t edit operations. All matching subgraphs are added into the result set. Algorithm 1 describes the above steps in pseudocode.

The algorithm starts from a query subgraph Q only containing n_q and a data subgraph g only containing n_g, and maps n_q to n_g. It iteratively adds edges from Q and G to Q and g respectively until Q is equal to Q and g is isomorphic to Q with at most t edit operations.

3.2 A Neighbourhood-based Pruning

In EXED, every node n_g of the data graph is considered a possible match of the query node n_q and as a seed to start the search for relevant answers.

Algorithm 1 EXED

```
Input:  Data graph $G = \langle V, E \rangle$, query graph $Q$
Input:  Threshold $t$
Output: Set of answers $S$
1: $S \leftarrow \emptyset$
2: $n_q \leftarrow chooseARandomNode(Q)$
3: for each node $n_g \in V$ do
4:     $s$ = SearchSimilarSubgraph($G$, $Q$, $n_q$, $n_g$, $t$)
5:     if $s \neq \emptyset$ then
6:         Add $s$ to answer set $S$.
7:     end if
8: end for
9: return $S$
```

This is highly inefficient since only a small fraction of data nodes are true candidates. To reduce this search space, one has to reduce the number of unnecessary data nodes from which the search for similar subgraphs starts. Inspired by [17], we propose a method called NeighbourhoodPruning to prune the search space.

Definition 6. *(d-neighbour) Let $n \in V$ be a node of the data graph $G = \langle V, E \rangle$. The node $n_i \in V$ is a d-neighbour of n if there exists a path from n to n_i of length at most d. The d-neighbourhood nodes of n, denoted as $N_d(n)$, is the set of all d-neighbours of n, and the d-neighbourhood labels of n, denoted as $L_d(n)$, is the set of edge labels on paths of length at most d from n to its d-neighbour nodes.*

NeighbourhoodPruning compares data nodes with query nodes using their neighbourhood information, and filters out those data nodes that requires more than t edit operations to match the query node's neighbourhood. Let $T_{n,k,l}$ denotes those neighbour nodes of a node n which are reachable from the node n in a path of length k and l is the last label in the path, i.e.

$$T_{n,k,l} = \left\{ n_1 | n_1 \xrightarrow{l} n_2 \vee n_2 \xleftarrow{l} n_1, n_2 \in N_{k-1}(n) \right\}.$$

where $n_1 \xrightarrow{l} n_2$ is an edge labeled with l.

Since keeping the table of neighbour nodes for every data node is expensive in term of space, we only keep the cardinality of $T_{n,k,l}$. Also, to efficiently retrieve candidates matching a query node, we implement an inverted index which stores a list of nodes for every label, every cardinality and every distance. In other words, the index allows us to efficiently find data nodes that have a label l at their k-neighbourhood with a certain cardinality.

Once the neighbourhood tables $T_{n,k,l}$ of both data and query nodes are computed for each label l and path length $k \leq d$ for some neighbourhood depth d, then we can compare the neighbourhood of a query node to that of a data node and filter out unqualified data nodes. We introduce the d-neighbourhood distance to filter out unqualified data nodes using neighbourhood table.

Definition 7. *(d-Neighbourhood Distance) The d-neighbourhood distance between a data node n_g and a query node n_q is the difference between their d-neighbourhood tables defined as*

$$dist(n_g, n_q) = \sum_{k=1}^{d} \sum_{l \in L_k(n_q)} M(T_{n_g,k,l}, T_{n_q,k,l}),$$

where $L_k(n_q)$ is the set of labels in the k^{th}-neighbourhood of n_q and M(x, y) is a positive difference function as given below:

$$M(x, y) = \begin{cases} 0 & \text{if } x \geq y \\ x - y & \text{otherwise.} \end{cases}$$

The reason to adapt a positive difference function is that if the subgraphs in the data graph carries more labels than the query, we shall not penalize it. Then, the d-neighbourhood distance between a data node n_g and a query node n_q for label l at k^{th}-neighbourhood can be written as

$$M(T_{n_g,k,l}, T_{n_q,k,l}) = \begin{cases} 0 & \text{if } |T_{n_g,k,l}| \geq |T_{n_q,k,l}| \\ |T_{n_g,k,l}| - |T_{n_q,k,l}| & \text{otherwise.} \end{cases}$$

A data node, which is at t neighbourhood distance from the query node, needs at least t label substitutions to be qualified as a candidate for the query node. Therefore, given an edit distance threshold t, n_g is considered a candidate for the query node n_q if the distance between the d-neighbourhoods of the two nodes does not exceed t, i.e.

$$dist(n_g, n_q) \leq t.$$

Note that this filtering may introduce false positives, because neighbourhood-based pruning cannot identify if the labels are in the same path. For example, the neighbourhood-based distance between q_1 and g_1 in Figure 3.1 is 0 whereas the actual edit distance is 6. It may be noted that the more correlated the edge labels in a query's path are, the less false positives the neighbourhood-based pruning can produce. This summarized representation of a neighbourhood is highly effective at pruning nodes without actually visiting their neighbourhood. False positives can be removed at the verification stage which takes the previous comparisons of the nodes into consideration.

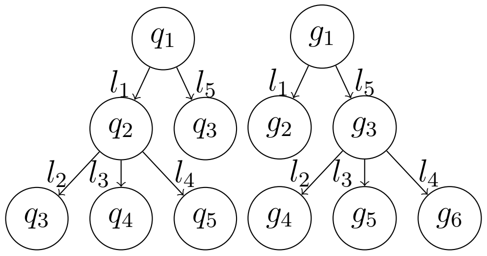


Figure 3.1: Query graph and data graph

Definition 8. *(Simulation) Let $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ be two graphs. G_2 simulates G_1 if there exists a relation $R \subseteq V_1 \times V_2$ such that, for every node $n_1 \in V_1$ and $n_2 \in V_2$ for which $(n_1, n_2) \in R$ and $n_1 \xrightarrow{l} n'_1$, there exists a $n'_2 \in V_2$ such that $n_2 \xrightarrow{l} n'_2$ and $(n'_1, n'_2) \in R$.*

While verifying a simulation, $n'_2 \in V_g$ is a possible match of $n'_1 \in V_q$ only if in a previous comparison of nodes n_2 and n_1, n_2 is identified as a possible match of n_1 and there is an edge between n_2 and n'_2 with label l and a corresponding edge with label l between n_1 and n'_1. With this observation, we only need to examine adjacent nodes of previously matched data nodes rather than all data nodes to find possible matches of a query node.

The EXED algorithm randomly chooses a query node n_q as a seed (starting node) and starts the search from the seed node. However, when we take the previous comparisons of the nodes into consideration, the choice of the starting node can affect the performance of the algorithm. The fewer previously matched data nodes are, the less comparisons we need do in the following steps of the simulation. We capture this by introducing the selectivity of query nodes and labels into our algorithm.

Definition 9. *(Selectivity) The selectivity of a query node n in a data graph G is the probability that a node of G matches n. The selectivity of a label l is the probability that an arbitrary edge of G is labeled l, and is computed as the ratio of the frequency of label l to the number of edges in G.*

As the actual selectivity of a query node may be known only after finding its matches, we devise a method to estimate the selectivity in advance (see Sec. 4.1 for details). In this section, we assume the selectivities of nodes are known.

Algorithm 2 NeighbourhoodPruning

```
Input:  Data graph $G = \langle V_g, E_g \rangle$, query graph $Q = \langle V_q, E_q \rangle$
Input:  Threshold $t$
Output: Set of candidate mappings $\mu \subseteq V_q \times N_g$
 1: $L_d^q \leftarrow d$-neighbour labels of $Q$
 2: Vis $\leftarrow \emptyset$
 3: $n_{min} \leftarrow \operatorname*{argmin}_{n \in V_g} Sel(n)$
 4: $N_g \leftarrow (V_g, 0)$
 5: $\mu(n_{min}) \leftarrow N_g$
 6: $Q \leftarrow \{n_{min}\}$
 7: for each $n_q \in Q$ do
 8:     if $n_q \xleftarrow{l} n'_q \in E_q \vee n_q \xrightarrow{l} n'_q \in E_q$ and $n'_q \notin$ Vis then
 9:         Update edit distance of nodes in $\mu(n_q)$, $\mu(n'_q)$.
10:         Remove nodes that exceed threshold.
11:     end if
12:     $Q \leftarrow Q \cup \{n'_q | n_q \xleftarrow{l} n'_q \vee n_q \xrightarrow{l} n'_q\}$
13:     $Q \leftarrow Q \setminus \{n_q\}$
14:     Vis $\leftarrow$ Vis $\cup \{n_q\}$
15: end for
```

Let n_{min} be a query node with the minimum selectivity. Our Algorithm 2 initially takes the set of all data nodes as candidate mappings of n_{min}. For each query node n_q that has not been visited yet, the algorithm checks if each data node $n \in V_g$ has the matching edges (i.e. edges with the same label and direction) for each adjacent edges of n_q. If it does not match and the edit distance t has already reached the threshold, (n, t) is removed from the mapping $\mu(n_q)$, where n is a possible candidate for query node n_q and t is the edit distance between the partial query containing n_q and partial matching containing n. If it does not match and t has not reached the threshold, a node n' adjacent to n is considered as a candidate for the query node n'_q adjacent to n_q and the entry $(n', t+1)$ is inserted into $\mu(n'_q)$. Otherwise, n' is a candidate match of n'_q with no edit penalty and the entry (n', t) is inserted into $\mu(n'_q)$. Finally, the query node n_q is marked as visited and is removed.

3.3 Improving Neighbourhood-based Pruning

The neighbourhood-based filtering may introduce false positives, because two matching labels may not be under the same path or have the same direction. In this section, we introduce a path-based filtering algorithm to prune out some of the false positives.

The path-based filtering algorithm compares data nodes with the query node in terms of their paths and filters out those data nodes that require more than t edit operations to match the query node. However, keeping every path for every node can be very expensive in term of space. For a graph with average degree $\hat{D}$ and d-edge path indexes, the space required for using path indexes is $O(\hat{D}^d)$. Our approach to reduce the space is using the Bloom filter[3].

A Bloom filter is a space-efficient probabilistic data structure to efficiently test whether an element is a member of a set N. An empty Bloom filter is a bit array of m bits, all set to 0. There are k different hash functions, each mapping an element to one of the m array positions. To query for an element, one needs to find the k array positions the element is mapped to. If any of the bits at these positions is 0, the element is definitely not in the set. If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive. The error rate p depends on m, $|N|$ and k. We set the false positive rate to 1%. The optimal number of hash functions is approximately $0.7m/|N|$, and the optimal number of bits m is approximately $|N| \ln p / \ln^2 2$. The number of inserted elements can be estimated by $\hat{D}^d$, where $\hat{D}$ is the average degree of the data graph[6]. The Bloom filter based path filtering allows us to control the false positives at a low rate with a compact storage and an efficient access time. Moreover, it has no false negatives.

To insert a path into the Bloom filter, we concatenate the labels in the path to form a string that is inserted into the Bloom filter. To encode the direction of an edge, a sign symbol is added to each label to distinguish between incoming and outgoing edges. In addition, the count of each path is described by preceding the label sequence and separated from the rest of string by “P”. For example, the string “2P+1-2” describes two paths that have one outgoing edge labeled with value 1 and one incoming edge labeled with value 2. Since all labels in a path are encoded into one string, an unmatched path can have up to d unmatched labels. To avoid filtering out false negatives, we consider the lower bound of the edit distance for an unmatched path, which is 1. This also introduces false positives if we only use path filtering. However, these false positives can be removed by considering the neighbourhood filter.

Our experiments show that the two filtering schemes work nicely, complementing each other. Path filtering can identify if multiple labels are in the same path and if the matching edges with the same labels have the same direction which neighbourhood filter cannot do; on the other hand, neighbourhood filtering can identify the level of mismatched labels which cannot be done by a path-based filtering.

3.4 Wildcard Approach

The main problem with EXED is that the number of intermediate results can become huge, especially for large edit distance thresholds and large node degrees of the data graph. Most of those intermediate results need to be kept until a very late stage of the searching.

To reduce the number of intermediate results, we develop a new algorithm referred to as wildcard queries with edit distance constraint (WCED). The approach taken in this algorithm is to map the subgraph edit distance problem instance into subgraph isomorphism problem instances without missing any relevant answers. This is done by introducing wildcard labels. A *wildcard label* is a label that can substitute for any other label in graph matching.

The main idea is to perform multiple subgraph isomorphism searches based on the original query and merge the retrieved answers to obtain the final results. This approach has two phases: query pre-processing and subgraph search and answer mergence.

In the query preprocessing phase, we choose t edges from $|E_q|$ query edges, where t is the edit distance threshold and set their labels to the wildcard for edge label substitution (see our discussion in this section for other edit operations). This gives us $O(\binom{|E_q|}{t})$ wildcard queries assuming that $t \leq |E_q|$. For example, Figure 3.2 shows a two-edge query and its wildcard queries with edit distance threshold 1.

In the next phase, we run subgraph isomorphism searches on those generated wildcard queries. For this, we directly adopt EXED with edit threshold set to 0. This returns the subgraphs where the wildcard matches any label. For example, searching for the first wildcard query in Figure 3.2 will give us all subgraphs which have an edge labelled l_1 and and an edge with any label, both under the same parent node. Finally, duplicates due to possible overlaps between wildcard queries are removed.

The WCED algorithm reduces the number of intermediate results by converting the subgraph edit distance into subgraph isomorphism. This is for the cost of running EXED $\binom{|E_q|}{t}$ times with edit distance threshold 0.

Other edit operations Supporting edge deletion is similar to substitution except that the edges are removed instead of being labeled with a wildcard. The only exception is that deleting an edge can result in a disconnected query graph, hence deletion may be applied to a subset of the edges whereas substitution can be applied to all edges. With $|E_q|$ query edges and edit distance threshold t, there are at most $\binom{|E_q|}{t}$ possible choices for deletion, each leading to a subgraph isomorphism search. As discussed in Section 1.3, edge insertion does not arise in Exemplar queries since we are searching for subgraphs of the query graph, and insertions are already supported at no cost. For example, the data graph can have any number of additional edges, and those edges are ignored in a subgraph search.

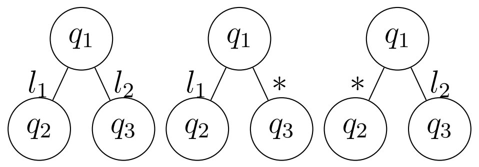


Figure 3.2: Query graph and its wildcard queries

Chapter 4

Algorithm Cost Analysis

We have presented two algorithms for exemplar queries with edit distance constraints, each with some advantages over the other. WCED has fewer intermediate results (meaning less space usage), while EXED only needs to be run once and has no duplicate answers. To determine which algorithm has the least cost for a given query and data graph (without actually running the algorithms), one needs an accurate cost estimation. This is the problem addressed in this section.

EXED consists of three parts: starting node selection, neighbourhood-based pruning and subgraph verification. The time cost of starting node selection and neighbourhood-based pruning are linear in the number of query nodes and number of data graph nodes respectively, while the time cost of subgraph verification grows exponentially with the edit distance threshold and the number of query edges. WCED consists of three phases: query pre-processing, subgraph isomorphism search and answer mergence. Subgraph isomorphism search uses EXED with the edit distance threshold 0, the cost of which also grows exponentially with the number of query edges. The time cost of query pre-processing depends on the number of query edges and the edit distance threshold. The time cost of answer mergence is linear in the number of answers. Both of them are relatively low and negligible compared to the cost of subgraph isomorphism search. Therefore, we focus on the cost of verification of two algorithms. The cost depends on the number of data nodes (candidates) matching the query starting node and the cost of verifying each candidate.

In this section, we first present an estimation for the selectivities of edge labels. We then present an exact cost model and an upper-bound cost model.

4.1 Selectivity Estimation

The probability that an arbitrary edge in the data graph has label l, referred to as the selectivity of label l, can be written as

$$Sel(l) = \frac{freq(l)}{|E_g|},$$

where $freq(l)$ is the frequency of label l in the data graph G, and $|E_g|$ is the number of edges in G.

4.2 An Exact Cost Model

Both algorithms EXED and WCED start with a set of candidate nodes in data graph G that are likely to match a query node n_s; those candidates may be selected based on a filtering scheme such as the neighbourhood or the path filtering. Given a candidate node in G, we must check if there is a subgraph in G that simulates the query graph in which the candidate node matches n_s. The cost of this process depends on two factors: the number of candidates matching the query node and the cost of verifying each candidate.

4.2.1 A cost model for WCED

Given a query and an edit distance threshold that is larger than zero, the WCED algorithm generates a set of wildcard queries based on the edit distance threshold, hence it has to perform multiple subgraph isomorphism searches on those wildcard queries. The cost is the sum of the costs of those searches. It should be noted that a wildcard query is like any query except that some edges are labeled with wildcards and those wildcards can match any label.

Estimating the number of candidates: Given a seed n_q, we want to estimate the probability that a data node is a candidate for n_q.

Lemma 1. *Given a query node and its adjacent edge labels $l_1, \dots, l_k$, and assuming independence between the labels, the probability that a data node* *with D adjacent labels has all query labels is*

$$P_D(l_1, l_2, ..., l_k) = \tag{4.1}$$
$$\sum_{j=1}^{k-1} \sum_{i=j+1}^{k} (-1)^{i-1} P_D(\neg l_j, \dots, \neg l_{i-1}, l_{i+1}, \dots, l_k) +$$
$$\sum_{j=1}^{k-1} (-1)^{k-j+1} (1 - \sum_{i=j}^{k} Sel(l_i))^D + (1 - (1 - Sel(l_k))^D.$$

Proof. This lemma can be proved using the probability subtraction rule:

$$\begin{aligned}
&P_D(l_1, l_2, ..., l_k) = \\
&P_D(l_2, ..., l_k) - P_D(\neg l_1, l_2, ..., l_k) = \\
&P_D(l_2, ..., l_k) - (P_D(\neg l_1, l_3, \dots, l_k) - P_D(\neg l_1, \neg l_2, ..., l_k)) = \\
&P_D(l_2, l_3, ..., l_k) - P_D(\neg l_1, l_3, \dots, l_k) + P_D(\neg l_1, \neg l_2, l_4, \dots, l_k) \\
&- P_D(\neg l_1, \neg l_2, \neg l_3, ..., l_k) = \\
&... \\
&\sum_{i=2}^{k} (-1)^{i-1} P_D(\neg l_i, \dots, \neg l_{i-1}, l_{i+1}, \dots, l_k) \\
&+ (-1)^k P_D(\neg l_1, \neg l_2, ..., \neg l_k) + P_D(l_2, l_3, ..., l_k).
\end{aligned} \tag{4.2}$$

If we expand $P_D(l_2, \dots, l_k)$ further using the equation above, we will have a set of terms that look similar to the first and the second terms in Eq 4.2 and the base case $P_D(l_k)$. For the base case, we have $P_D(l_k) = (1-(1-Sel(l_k))^D$. We also know that $P_D(\neg l_j, \dots, \neg l_k) = (1-\sum_{i=j}^{k} Sel(l_i))^D$ assuming independence. Putting these pieces together will give the statement of the lemma.

□

Lemma 1 directly gives the selectivity of a query node based on its 1-neighbourhood. Let $L_i(n_q)$ denote the set of labels at the i^{th} neighbourhood of a query node n_q. The probability that the neighbourhood of a data node matches that of a query node at levels $1, \dots, d$ can be written as

$$P(n_q) = \prod_{m=1}^{d} P_{D_m}(L_m(n_q)). \tag{4.3}$$

where $P_{D_m}(L_m(n_q))$ is as defined in Lemma 1 and D_m is the number of edges at the m^{th} neighbourhood of a data node. We generally don't know D_m when estimating our probabilities in Equations 4.1. Assuming that each data node has the same degree $\hat{D}$, then $D_m = \hat{D}^m$. Let $C(n_q)$ denote the candidate set for query node n_q. We can have the number of candidates matching query node n_q as

$$|C(n_q)| = |V_g| * P(n_q).$$

Estimating the cost of verifying each candidate: For each candidate of the starting node, the algorithm starts from a graph g with only one node (i.e. the candidate node) and iteratively adds new edges to g until either g simulates the query, or no such simulation is found. The cost of adding each new edge depends on the expected number of matching edges of a query edge and the number of subgraphs to which the edges are added. Let $\hat{D}$ denote the expected degree of a data node. For a query label l_i, we expect $\hat{D} * Sel(l_i)$ edges of a node in the data graph to match l_i. For a fixed candidate node in the data graph, the expected number of subgraphs (partial matches) that can be constructed starting from the candidate and simulating the query subgraph rooted at the seed with labels $l_1, \ldots, l_k$ is

$$\prod_{i=1}^{k} \hat{D} * Sel(l_i).$$

and the total expected cost of verifying a candidate n is

$$\sum_{i=1}^{|E_q|} \prod_{j=1}^{i} \hat{D} * Sel(l_j). \tag{4.4}$$

Note that this is based on the assumption that a search starting from a candidate node will not stop early if the simulation exceeds the edit distance threshold. The total cost of verifying $|C(n_q)|$ candidates is

$$Cost(q) = |C(n_q)| * \sum_{i=1}^{|E_q|} \prod_{j=1}^{i} \hat{D} * Sel(l_j). \tag{4.5}$$

Since we have replaced a query graph with $\binom{|E_q|}{t}$ graphs each with t wildcards, the total cost is the sum of the costs of verifying those wildcard queries.

4.2.2 EXED Cost Model

To estimate the cost for EXED, we also need to estimate the number of candidates in the data graph matching a query seed node and the cost of verifying each candidate.

Estimating the number of candidates: Since a data node is allowed to have up to t edit operations in its neighbourhood, directly estimating the probability that a data node is a qualified candidate is difficult. Therefore, we estimate the number of candidates for a set of wildcard queries where the labels are all fixed. By summing up the number of candidates for these wildcard queries and removing the repetitive candidates due to overlaps between queries, the number of candidates for n_q in EXED can be written as

$$\begin{aligned} |C(n_q)| = &\sum_{i=1}^{\binom{|E_q|}{t}} |V_g| * P(n_{w_i(q,t)}) \\ &- (\binom{|E_q|}{t} - 1) * |V_g| * P(n_q). \end{aligned} \tag{4.6}$$

where $w_i(q, t)$ is a wildcard query constructed from q by replacing t edge labels with wildcards and $P(n_q)$ is as in Equation 4.3. The last term gives the number of double-count candidates for $\binom{|E_q|}{t}$ wildcard queries.

Estimating the cost of verifying each candidate: To estimate the cost of verifying each candidate, we need to estimate the number of partial matches. There are two kinds of partial matches in EXED: (1) matches that have reached the edit distance threshold, and (2) matches that have not reached the threshold. For (1), edges with any label can be added to the matching in the next step of the simulation, whereas for (2), only edges with matching labels can be added. Let m be the number of edges in a partial matching, and k be the number edges in the matching where the matching edges have different labels. If $l_1, \ldots, l_k$ denote the query labels in the matching where the labels don't match, and $l_{k+1}, \ldots, l_m$ be the labels where both data and query edges in the matching have the same labels, then the number of partial matches can be written as

$$\hat{D}^m \prod_{i=1}^{m-k} Sel(l_i) \prod_{j=1}^{k} (1 - Sel(l_j)).$$

Given query labels $l_1, \ldots, l_m$, we generally don't know in advance which labels will mismatch and need to check all choices of $\binom{m}{t}$ sets of labels. The number of partial matches that needs to be verified is

$$S_t(q,m) = \begin{cases} 0 & \text{if } t > m \\ \hat{D}^m \prod_{i=1}^{m} Sel(l_i) & \text{if } t = 0 \\ \sum_{k=1}^{\binom{m}{t}} \hat{D}^m \prod_{i=1}^{m-t} Sel(l_{k,i}) \prod_{j=1}^{t} (1 - Sel(l_{k,j})) & \text{if } t < m. \end{cases} \tag{4.7}$$

For any partial matching that have not reached the threshold t, any edge can be added into the matching in the next step of the simulation. In this case, the next step of simulation costs

$$\sum_{j=0}^{t-1} S_j(q,m) * \hat{D}.$$

For any partial matching that have reached the threshold, only edges with a matching label can be added. In this case, the next step of a simulation costs

$$S_t(q,m) * \hat{D} * Sel(l_{m+1}).$$

The cost of verifying each candidate in EXED is

$$Cost(q) = \sum_{i=0}^{|E_q|-1} (S_t(q,i) * \hat{D} * Sel(l_{i+1}) + \sum_{j=0}^{t-1} S_j(q,i) * \hat{D}), \tag{4.8}$$

and the total cost of EXED is the product of the number of candidates (as given in Equation 4.6) and the cost of verifying a candidate (as given above).

$$Cost_{ex} = |C(n_q)| Cost(q).$$

4.2.3 Cost Model Comparison

We want to compare the costs of verifying the candidates for EXED and WCED and identify the conditions under which one outperforms the other. Our cost comparison assumes that the threshold t is less than the number of query edges; otherwise, edges in the query can match every edge in the data graph, the labels become irrelevant and the problems becomes subgraph isomorphism on unlabeled graphs, which has been studied in many other works and we will not address this problem in this thesis.

For the edit distance threshold larger than zero, the cost of verifying a candidate in EXED is higher than that in WCED, because edit operations can happen on any label in EXED while they are fixed in WCED. Hence if the number of candidates for WCED and EXED are roughly the same, WCED will outperform EXED. In other words, WCED outperforms EXED if the number of candidates for the original query is small (See Equation 4.6). This is a more plausible scenario for our queries; otherwise edit operations are less likely to be considered. The next lemma shows what happens when this condition does not hold.

Lemma 2. *Given a data graph with expected node degree $\hat{D}$, a query graph Q with at least 2 edges and the edit distance threshold set to 1, the cost of verifying a candidate in EXED is less than the sum of the cost of verifying a candidate for every wildcard queries in WCED when*

$$Sel(l_1) > \frac{1}{\sqrt[|E_q|]{\hat{D}}}. \tag{4.9}$$

where l_1 is a query label that has the highest selectivity (i.e. the smallest value of $Sel(l_i)$).

Proof. Using Equation 4.4, the cost of verifying each wildcard query for WCED with edit distance 1 can be written as

$$Cost_{wc} = \sum_{k=1}^{|E_q|} \sum_{i=1}^{|E_q|} \prod_{j=1}^{i} \hat{D} * Sel(l_{k,j}).$$

Let $l_1, \ldots, l_k$ denote the labels in increasing order of selectivities. Since the edges in a query are verified in increasing order of selectivities, for those wildcard queries where $l_j (1 \leq j \leq i)$ is not set to wildcard, the edge with label l_i is verified at i^{th} step of the simulation, the cost of verifying the edge is $\hat{D}^i \prod_{j=1}^{i} Sel(l_j)$, there are $|E_q| - i$ these type of wildcard queries; for those that $l_j (1 \leq j \leq i)$ is set to wildcard, the edge with label l_{i+1} is verified at i^{th} step of the simulation, the cost of verifying this edge is $\hat{D}^i \prod_{j=1}^{i+1} Sel(l_j)$, where $l_j \neq l_m$.

Let T_i be

$$T_i = \sum_{k=1}^{i} \prod_{m=1}^{i} Sel(l_{k,m}) \text{ where } l_{k,m} \neq l_k, \tag{4.10}$$

which will be used in following proof to simplify the other equations.

The sum of verifying costs for those wildcard queries that $l_m (1 \leq m \leq i-1)$ is set to wildcard at i^{th} step of the simulation is $\hat{D}^i(T_{i+1} - \hat{D}^i \prod_{j=1}^{i-1} Sel(l_j))$. Then, the sum of verifying costs for WCED can be written as

$$\begin{aligned} Cost_{wc} = &\sum_{i=1}^{|E_q|-1} \hat{D}^i((|E_q| - i - 1) \prod_{j=1}^{i} Sel(l_j) + T_{i+1}) \\ &+ \hat{D}^{|E_q|} T_{|E_q|}. \end{aligned}$$

Using Equations 4.7 and 4.8, the verifying cost of EXED with edit distance threshold 1 can be written as

$$\begin{aligned} Cost_{ex} = \hat{D} + &\sum_{i=2}^{|E_q|} (\hat{D}^i \sum_{k=1}^{i-1} (1 - Sel(l_k)) \prod_{j=1}^{i} Sel(l_{k,j}) \\ &+ \hat{D}^i \prod_{j=1}^{i-1} Sel(l_j)), \text{where } l_{k,j} \neq l_k. \end{aligned}$$

Using T_i in Equation 4.10 to simplify above equation, $Cost_{ex}$ can be written as

$$Cost_{ex} = \hat{D} + \sum_{i=2}^{|E_q|} \hat{D}^i (T_i - (i-1) \prod_{j=1}^{i} Sel(l_j)).$$

Then, combining the costs of WCED and EXED and computing the difference between the costs of EXED and WCED will give us

$$\begin{aligned} \Delta_{cost} = &\hat{D}(1 - (|E_q| - 1) Sel(l_1) - Sel(l_2)) \\ &+ \sum_{i=2}^{|E_q|-1} \hat{D}^i (T_i - T_{i+1} - (|E_q| - 2) \prod_{j=1}^{i} Sel(l_j)) \\ &- (|E_q| - 1) \hat{D}^{|E_q|} \prod_{i=1}^{|E_q|} Sel(l_i). \end{aligned}$$

Since query edges are visited in increasing order of label selectivities and the value of selectivities is less than 1, we have an inequality as follows

$$Sel(l_1) \leq Sel(l_i) \leq 1.$$

With the inequality above, we have

$$iSel(l_1)^{i-1} \le T_i = \sum_{k=1}^{i} \prod_{m=1}^{i} Sel(l_{k,m}) \le i.$$

Using both inequalities above, the upper bound of Δ_{cost} can be written as

$$\begin{aligned}\Delta_{cost} &\le \hat{D}(1 - |E_q|Sel(l_1)) - (|E_q| - 1)\hat{D}^{|E_q|}Sel(l_1)^{|E_q|} \\ &+ \sum_{i=2}^{|E_q|-1} \hat{D}^i(i - (|E_q| + i - 1)Sel(l_1)^i).\end{aligned}$$

Let $F_n(x)$ denote the upper bound of Δ_{cost}, x denote $Sel(l_1)$ and n to denote the number of query edges. To show the correctness of the Lemma 2, we only need to prove that $F_{|E_q|}(x) \le 0$ with different number of edges when the conditions in the Lemma holds using mathematical induction.

Basis: $n = 2$: $F_2(x)$ can be written as

$$F_2(x) = \hat{D}(1 - 2x) - \hat{D}^2x^2.$$

When $x = \frac{1}{\sqrt{\hat{D}}}$, we have

$$F_2(x) = \hat{D}(1 - 2x) - \hat{D}^2(\frac{1}{\sqrt{\hat{D}}})^2 = \hat{D}(-2x) < 0.$$

We also know that the derivative of $F_2(x)$ is

$$\frac{\partial F_2}{\partial x} = -2\hat{D} - 2\hat{D}^2x < 0.$$

Combining two facts above, we know that $F_2(x) < 0$ when $Sel(l_1) > \frac{1}{\sqrt{\hat{D}}}$.

Induction hypothesis: Assume the Lemma holds when the query has k edges.

$$\begin{aligned}F_k(x) = \hat{D}(1 - kx) - (k - 1)\hat{D}^kx^k \\ + \sum_{i=2}^{k-1} \hat{D}^i(i - (k + i - 1)x^i) < 0 \\ \text{subject to } x > \frac{1}{\sqrt[k]{\hat{D}}}.\end{aligned}$$

Note that the derivative of $F_k(x)$ is

$$\frac{\partial F_k}{\partial x} = -k - k(k - 1)\hat{D}^kx^{k-1} - \sum_{i=2}^{k} i(k + i - 1)x^{i-1} < 0.$$

Induction: Using $F_k(x)$ to substitute some terms in $F_{k+1}(x)$, $F_{k+1}(x)$ can be written as

$$F_{k+1}(x) = \hat{D}(1-(k+1)x) - k\hat{D}^{k+1}x^{k+1}$$
$$+\sum_{i=2}^{k}\hat{D}^i(i-(k+i)x^i) =$$
$$F_k(x) - \hat{D}x - \sum_{i=2}^{k-1}x^i - \hat{D}^k(k-(k+1)x^k) - k\hat{D}^{k+1}x^{k+1}.$$

When $x = \frac{1}{\sqrt[k+1]{\hat{D}}}$, after replacing the x with the value in the last term and combining the last two terms, $F_{k+1}(x)$ can be written as

$$F_{k+1}(\frac{1}{\sqrt[k+1]{\hat{D}}}) = F_k(x) - \hat{D}x - \sum_{i=2}^{k-1}x^i + \hat{D}^k(k-(k-1)x^k)$$
$$-k\hat{D}^{k+1}(\frac{1}{\sqrt[k+1]{\hat{D}}})^{k+1} = F_k(x) - \hat{D}x - \sum_{i=2}^{k-1}x^i - (k+1)\hat{D}^kx^k.$$

Since $x = \frac{1}{\sqrt[k+1]{\hat{D}}} > \frac{1}{\sqrt[k]{\hat{D}}}$, $F_k(x) < 0$ and the rest of terms are also negative, we have

$$F_{k+1}(\frac{1}{\sqrt[k+1]{\hat{D}}}) < 0.$$

We also know that the derivative of $F_{k+1}(x)$ is negative.

$$\frac{\partial F_{k+1}}{\partial x} = \frac{\partial F_k}{\partial x} - \hat{D} - \sum_{i=2}^{k-1}ix^{i-1} - k(k-1)\hat{D}^kx^{k-1} < 0.$$

Combining two facts above, we know that $F_{k+1}(x) < 0$ when $Sel(l_1) > \frac{1}{\sqrt[k+1]{\hat{D}}}$.

□

When the number of candidates for the original query is large (more precisely, roughly equal to the number of candidates for a wildcard query), the cost of EXED and WCED can both be approximated based on the number of candidates for the original query. In this case, EXED can outperform WCED given the condition of the lemma.

4.3 An Upper Bound Cost Model

The exact cost model is based on two assumptions: (1) labels are evenly distributed, and (2) labels are pairwise independent. These assumptions may not hold in real-world data graphs. This is a problem especially for large queries since the error can accumulate and become significant as the number of query edges increases. In this section, we present a cost model that gives an upper bound of the actual cost but is more accurate for larger query graphs.

Estimating the number of candidates: To estimate the upper bound for the number of candidates, two weaker assumptions of label independence are considered: (1) the labels of the adjacent edges of a data node are independent whereas labels, which are in a path starting from a node, are correlated; (2) the labels of the adjacent edges of a data node are correlated whereas labels, which are in a path starting from the node, are independent. For two or more correlated labels, the selectivity of the label with the least selectivity provides an upper bound of the selectivity of the set.

Under the first assumption, the selectivity of the label with the minimum selectivity in each path is used to estimate the selectivity upper bound of the path. This reduces each path in the query to an edge (with the minimum selectivity), and as a result the query becomes a node with a set of adjacent edges (i.e. a tree with only one level). Assuming independence between the labels of these edges, Lemma 1 will give an upper bound of the probability that a data node is a candidate for a query node. Note that D in the Lemma is set to the number of paths in the d-neighbourhood.

Under the second assumption, all edges under a node are collapsed into a single edge, which is labeled with a label from the set that has the least selectivity. Since the edge labels of the resulting query are all independent, Equation 4.3 can be used to estimate the upper bound.

Estimating the cost of verifying each candidate: To estimate the upper bound for the cost of verifying each candidate, the maximum frequency of each label under a node is used to upper bound the number of matching label in each step of the simulation. Let $N(l_i)$ denote the maximum frequency of label

l_i in the adjacent edges of a node.

In our exact cost model, the number of matching labels for a label l_i is $\hat{D} * Sel(l_i)$ assuming that every label is uniformly distributed on the adjacent edges of a node. Replacing $\hat{D} * Sel(l_i)$ in Equations 4.5 and 4.8 by $N(l_i)$ will give us an upper bound of the cost of verifying each candidate in WCED and EXED respectively.

Chapter 5

Experimental Evaluation

This section presents an experimental evaluation of our algorithms and cost models. All our experiments were performed on a 2.4 GHz 8 Core CPU with 100G memory running Linux. The algorithms are implemented in Java 1.8. Unless explicitly stated otherwise, the path length d in our filtering scheme is set to 3.

Dataset: We downloaded a full dump of Freebase[1] in May 2015. We removed the triples that were used as internal specification for the community (e.g. user and group data and discussion topics) obtaining a fully connected graph of 84 million nodes and 335 million edges . Since the entire Freebase is too large for our machine (occupies approximately 90G of memory when fully loaded), we extract subgraphs from Freebase with different parameters. The subgraphs are extracted using a breadth first traversal of the graph from a randomly selected starting node and randomly choosing new edges to be included in the data graph. Unless explicitly stated otherwise, the data graphs are randomly generated from Freebase with the number of nodes set to 10K and average node degree set to 15.

Queries: Two types of queries are used in our experiments: (1) A set of real queries from the AOL query log, manually mapped to the data graph, and (2) randomly selected subgraphs of the data graph. These queries vary in the numbers of edges and the selectivities of their labels. Unless explicitly stated otherwise, our experiments use 100 randomly selected queries, each a subgraph

[1]https://developers.google.com/freebase/

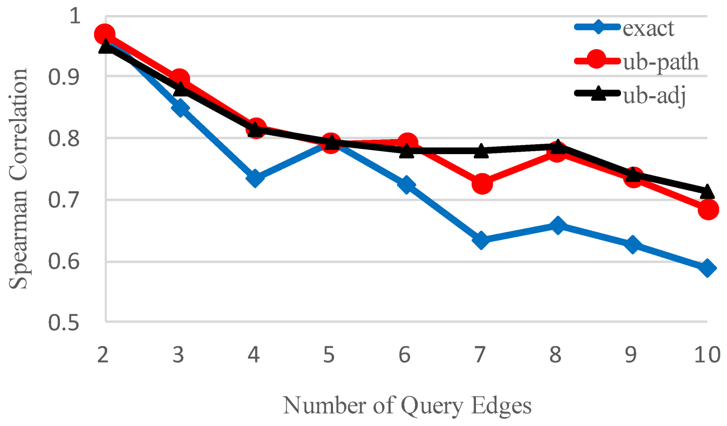
of the data graph.

Summary of our experiments: Our cost models are evaluated in Section 5.1, and the effectiveness of our filtering schemes under different settings and combinations is evaluated in Sections 5.2 and 5.3. The impact of our filtering schemes on the performance of our algorithms and improvements over existing algorithms are evaluated in Sec 5.4.

5.1 Effectiveness of Our Cost Models

In this section, we evaluate our cost models in terms of the correlation between our estimates and the actual costs. Our results show that: (1) the selectivity estimation is reliable when the number of query edges $|E_q| \leq 10$ (See Figure 5.1 and its discussions); (2) there is a linear relationship between our exact cost model and the real cost for $|E_q| \leq 3$, which allows us to estimate the running time of our algorithms (See Figure 5.2 and its discussions); (3) the exact cost is reliable for the comparison of our algorithms when $|E_q| \leq 6$ (See Figure 5.4 and its discussions); (4) the upper-bounded cost is reliable for the comparison of query costs when $|E_q| \leq 8$ (See Figure 5.3 and its discussions).

Effectiveness of the selectivity estimation Since selectivity estimation is a core component of our cost model, we first assess the quality of our selectivity estimation. To do so, we measure the correlation between the actual number of candidates and the estimated number of candidates based on our selectivity estimates. In our case, the selectivity is used in choosing a query starting node and for cost comparisons, hence, a relative ordering of the selectivity values is sufficient in theses cases. Therefore, we chose Spearman's rank correlation between estimate and actual selectivities, which shows the monotonic relationship of the two variables. The experiments are in the context of WCED and EXED algorithms. Let "exact" denote the exact selectivity estimation, "ub-path" and "ub-adj" denote the upper bound of the selectivity estimations respectively assuming that path labels and adjacent labels are independent. Figure 5.1 shows that although "exact" has a better correlation (0.96) than "ub-adj" and "ub-path" for queries with 2 edges, the correlation decreases rapidly for queries with a large number of edges, with about 0.55 correlation for queries with 10 edges. In contrast, the upper-bound selectivity estimations remain stable with different numbers of query edges in both WCED and EXED (with correlation between 0.7 and 0.96 and significance level between $3.08E - 64$ and $5.38E - 16$). We conclude that there is strong positive correlation between the selectivity estimation and the actual selectivity and that we can safely use this selectivity estimation for the choice of a query starting node and cost comparisons.



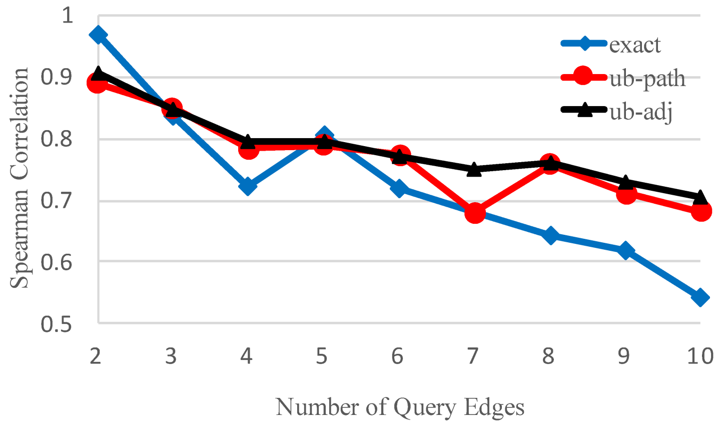


Figure 5.1: Correlation between estimated and actual selectivities for WCED (top) and EXED (bottom)

Exact cost model evaluation In this set of experiments, we examine the linear relationship between our estimated cost and the actual number of operations using Pearson correlation. The larger absolute value of the coefficient, the stronger the relationship between the actual cost and the estimated cost. If the absolute value of coefficient is large, with simple linear regression, we can predict the actual cost from our estimated cost. Figure 5.2 shows that for small queries (with up to 3 edges), the correlation coefficient is over 0.7 and 0.6 for WCED and EXED respectively. However, the correlation coefficient drops sharply as the number of edges increases. These results are expected because the exact cost model is based on the assumption that the labels are evenly distributed and that they are independent.

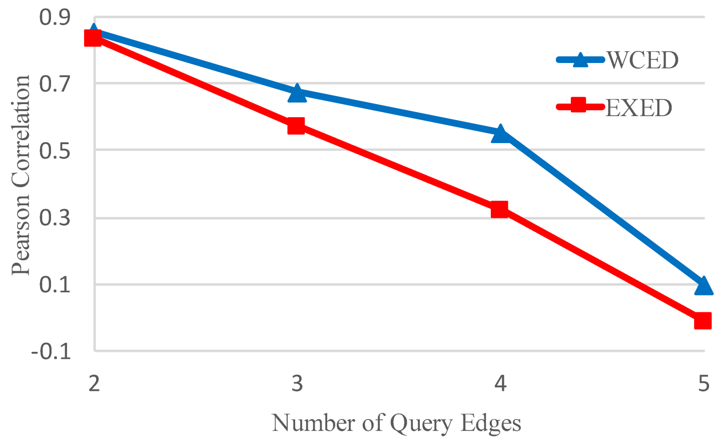


Figure 5.2: Correlation between estimated and actual costs varying the number of query edges

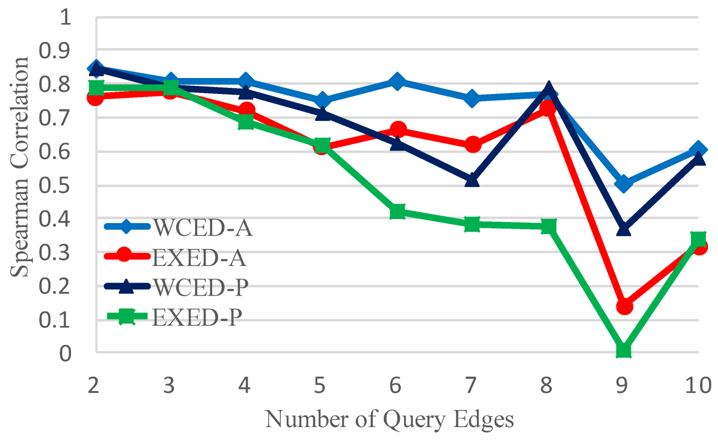


Figure 5.3: Correlation between estimated and actual costs varying the number of query edges

Upper-bound cost model evaluation In this set of experiments, we evaluate the upper bound cost models of our algorithms presented in Section 4.3. Let's denote with WCED-A and EXED-A the upper bounds of the cost model of WCED and EXED assuming independence of adjacent labels respectively, and denote with WCED-P and EXED-P the upper bounds of the cost model of WCED and EXED assuming independence of path labels respectively. Figure 5.3 shows that both WCED-A and EXED-A have a better Spearman correlation with the actual cost than both WCED-P and EXED-P. Both WCED-A and EXED-A have over 0.6 correlation for queries up to 8 edges, while WCED-P only has 0.5 correlation when queries have 7 edges and the correlation of EXED-P drops below 0.4 when queries have more than 6 edges. Based on these results, both WCED-A and EXED-A provide good cost models for comparing the cost of different queries.

Sometimes we have a query and want to find the algorithm that has the least cost before running the algorithms. To evaluate the effectiveness of our cost models, we computed the gaps between the actual costs of WCED and EXED, i.e. actW-actE and their estimated costs, i.e. estW-estE. A high correlation between the two gaps indicates that the cost model can show which algorithm has the least cost even though the actual value of the estimate may not be accurate. Figure 5.4 shows that for queries with up to 6 edges, the correlation coefficient is good (over 0.7).

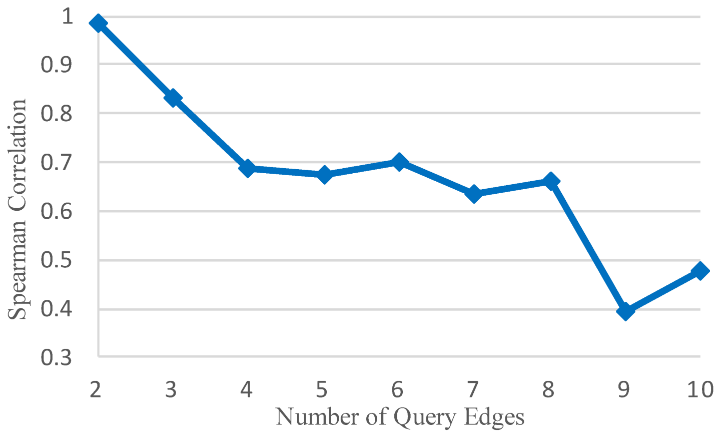
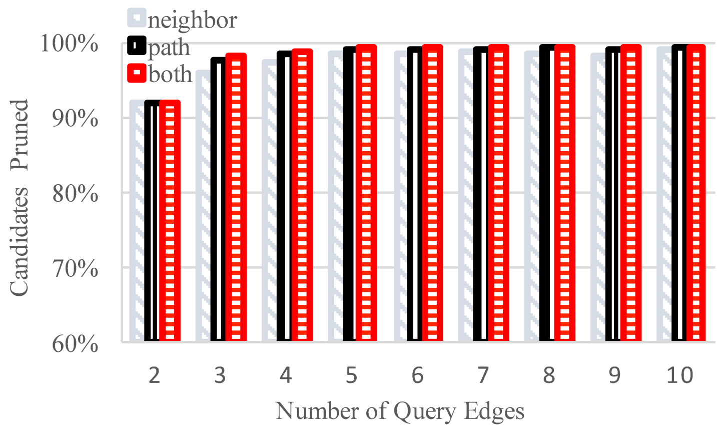


Figure 5.4: Correlation between actual and estimated cost differences varying the number of query edges

5.2 Effectiveness of Our Filtering Strategies

To evaluate the pruning power of our filtering schemes, the number of nodes in the data graph was set to 10K and the edit distance threshold was set to 1. Let "neighbour" denote the neighbourhood-based filtering strategy, "path" denote the path-based pruning strategy and "both" denote the case where both schemes were used.



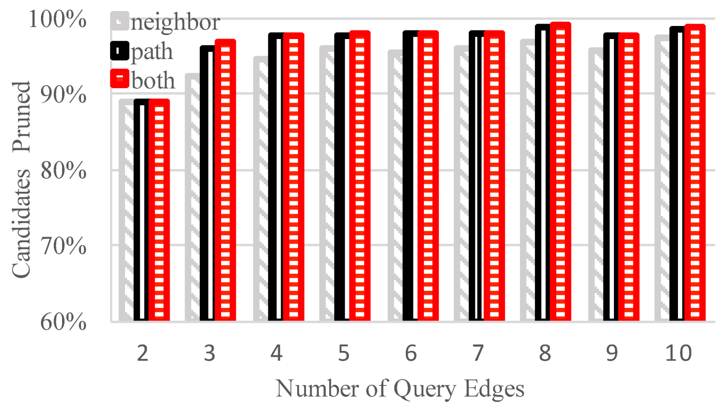
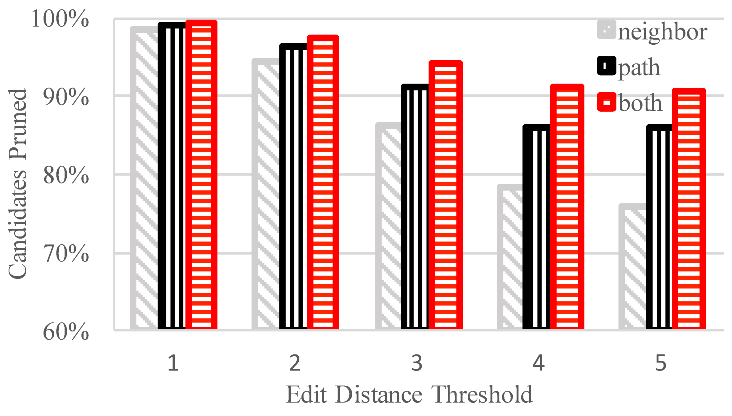


Figure 5.5: Pruning power of WCED (top) and EXED (bottom) varying the number of query edges

Varying the number of query edges: For this experiment, we varied the number of query edges from 2 to 10 and set the edit distance threshold to 1. Figure 5.5 shows the fraction of candidates that are pruned in EXED and WCED as we vary the number of query edges. As shown for WCED and

EXED, "path" can filter out respectively up to 99.4% and 99.1% of the data nodes on average, while "neighbour" can filter out respectively up to 99.0% and 97.3% of the data nodes on average. The pruning power does not increase by more than 1% when both strategies are used. However, considering the large number of data nodes and the high cost of verifying each candidate, even a small improvement in the pruning stages is amplified and positively affects the performance of the algorithms (See Figure 5.10).



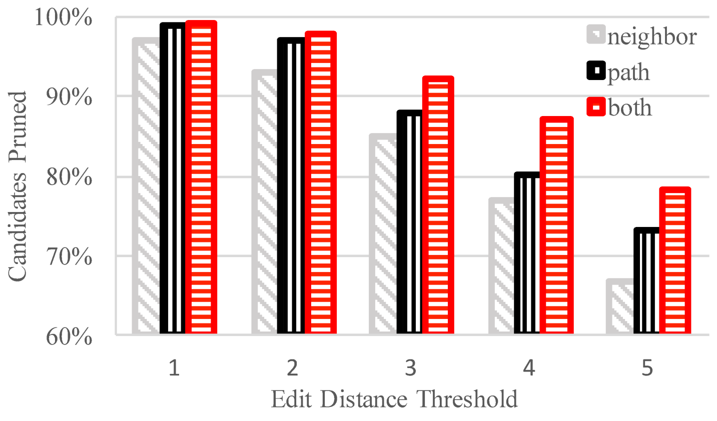
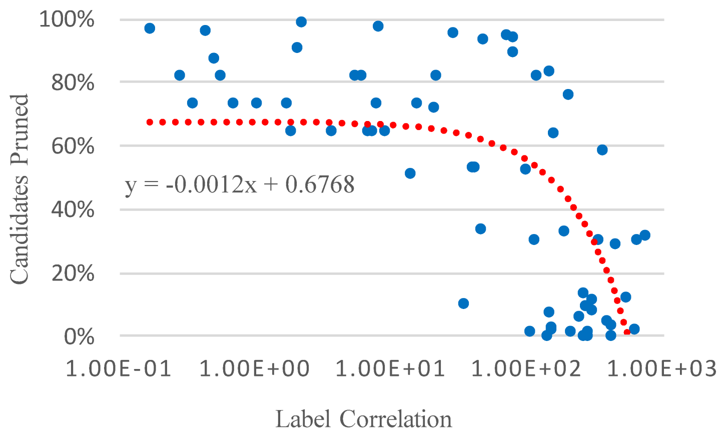


Figure 5.6: Pruning Power of WCED (top) and EXED (bottom) for different edit distance thresholds

Varying the edit distance threshold: For this experiment, the number of query edges was fixed at 8 with the edit distance threshold varied from 1 to 5. When the edit distance threshold is equal to or exceeds the number of query edges, edges in the query can match every edge in the data graph, the labels become irrelevant and the problems becomes subgraph isomorphism on unlabeled graphs, which is not the problem addressed in this thesis. Figure 5.6 shows that both "neighbour" and "path" have good pruning power (over 78%) under different distance thresholds, and it becomes more effective to apply both filtering schemes as the edit distance threshold increases. This is because "neighbour" scheme does not encode edge direction in its indexes and higher edit distance threshold introduces more false positives with wrong edge direction, while adding "path" on top of "neighbour" can effectively prune out those false positives.



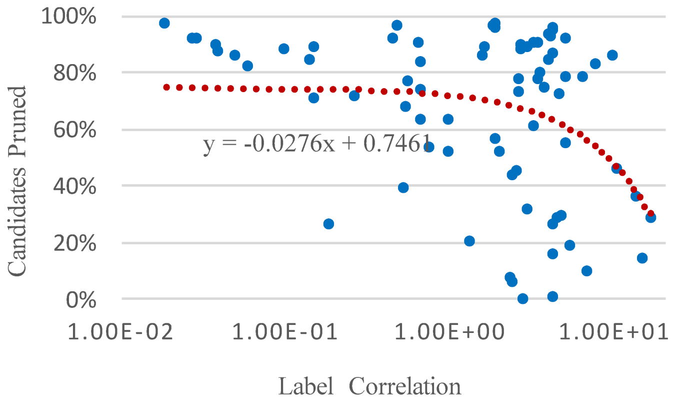


Figure 5.7: Candidates pruned vs path label correlation of WCED (on the top) and EXED (on the bottom)

Varying path label correlation: For this experiment, the neighbourhood filtering scheme is considered as a baseline, on top of which we added our path filtering scheme and monitored the improvement in pruning power. We fixed the number of query edges at 8 and set the edit distance threshold to 1. As shown in Figure 5.7 , the improvement in pruning power by adding "path" in both EXED and WCED drops with more correlation. This meets our expectation since the more correlated the labels are, the less false positives the neighbourhood-based pruning can produce and the less room for "path" filtering improvements.

5.3 Combining Filtering Schemes

In these set of experiments, we evaluate the impact of adding path filtering on top of the neighbourhood filtering. In order to show the impact of using both filtering schemes, we consider EXED with the neighbourhood filtering scheme as our baseline and compare it against EXED with both filtering schemes, WCED with the neighbourhood filtering scheme and WCED with both filtering schemes. We denote EXED and WCED with the neighbourhood filtering scheme as "neighbour-EXED" and "neighbour-WCED" respectively, WCED and EXED with both filtering schemes as "both-WCED" and "both-EXED" respectively.

Varying the edit distance threshold: We varied the edit distance threshold t from 1 to 5 and fixed the number of query edges at 8. Figure 5.8 shows that "neighbour-WCED" outperforms EXED by a factor of 1.5 when $t = 5$, reducing the search time more than half in this particular experiment. Comparing "neighbour-WCED" and "both-WCED", we find that even though there is no clear speedup for adding path filtering on top of the neighbourhood filtering scheme at small thresholds ($t \leq 2$), the performance gap becomes wider at larger thresholds with around 200 seconds saved when $t = 5$. This meets our expectation because the benefits of using both schemes over "neighbour" in pruning power becomes clear when the edit distance threshold increases (See Figure 5.6).

Varying average degree of data graph: In another experiment, we varied the average degree of a node from 5 to 25. Figure 5.9 shows that "both-WCED" has a greater advantage in a data graph with larger average degrees,

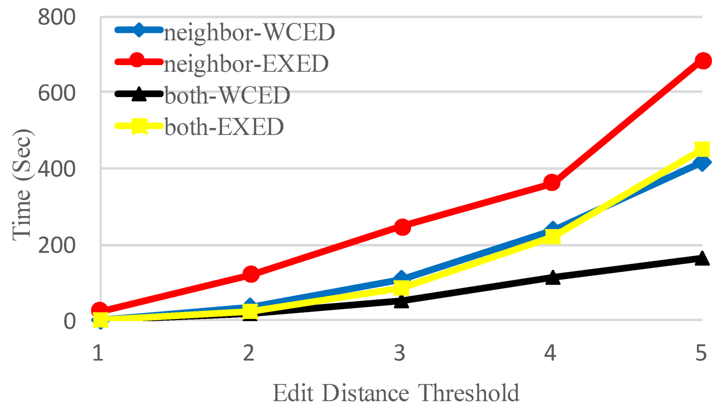


Figure 5.8: Time vs edit distance threshold

outperforming "neighbour-WCED" and "neighbour-EXED". This is because the cost of verifying each candidate depends on the average degree of the data graph, and a larger average degree results in a higher cost of verifying each candidate and thus a wider gap between "both-WCED" and the others.

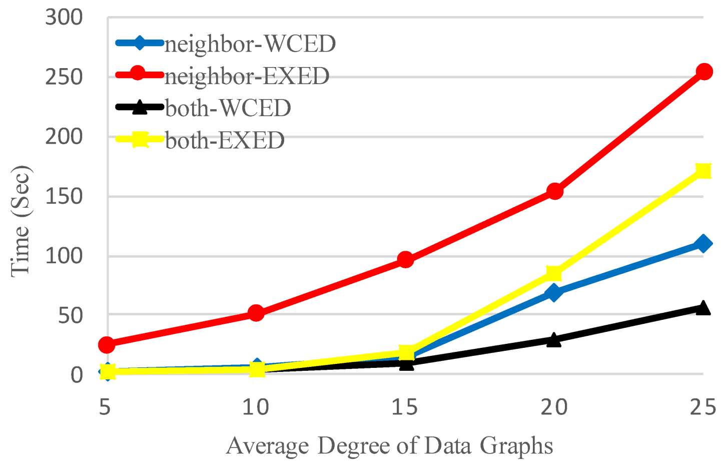


Figure 5.9: Time vs average degree of data graphs

Varying the number of query edges: In another experiment, we varied the number of query edges from 2 to 10 with the edit distance threshold fixed at 1. Figure 5.10 shows that the gap between "neighbour-WCED" and "both-WCED" (and similarly between "neighbor-EXED" and "both-EXED") widens as we increase the number of edges. This meets our expectation, because the cost of verifying each candidate grows exponentially with the number of edges.

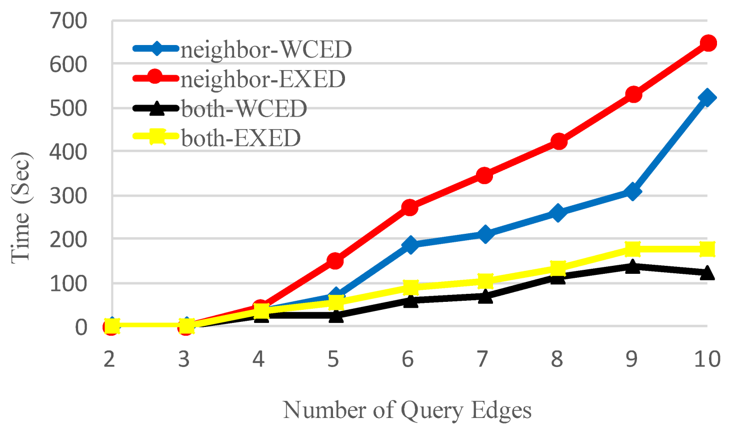


Figure 5.10: Time vs number of query edges

Our experiments in this section reveals that adding path filtering improves the performance of both algorithms EXED and WCED under one or more of these conditions: (1) the data graph has high average degree; (2) the edit distance threshold $t \geq 2$; (3) the query has over 5 edges.

5.4 Algorithms Comparison

To evaluate the performance of our algorithms against the competitors, we selected two recent algorithms from the literature: (1) SAPPER[43] which is an algorithm for indexing and approximate matching in large graphs, and (2) Exemplar[23] which is similar to our work but is limited to edit distance threshold zero.

Comparing against SAPPER: In this set of experiments, we compare the scalability of our algorithms against SAPPER. In the first experiment, we varied the number of nodes in the data graph from 10K to 1M while the edit distance threshold was set to 1. This is consistent with the settings by the authors of SAPPER except that their largest data graph had only 10K nodes. Since SAPPER only supports edge deletion (missing edges), we modified SAPPER to support edge label substitutions. Figure 5.11 shows the running time for retrieving the first 100 answers. The results show that SAPPER is up to 15 times slower than our algorithms, and becomes slower as the data graph size increases. In contrast, our algorithms are not much sensitive to the size of the data graph and scale gracefully to large data graphs, exhibiting an almost constant behaviour.

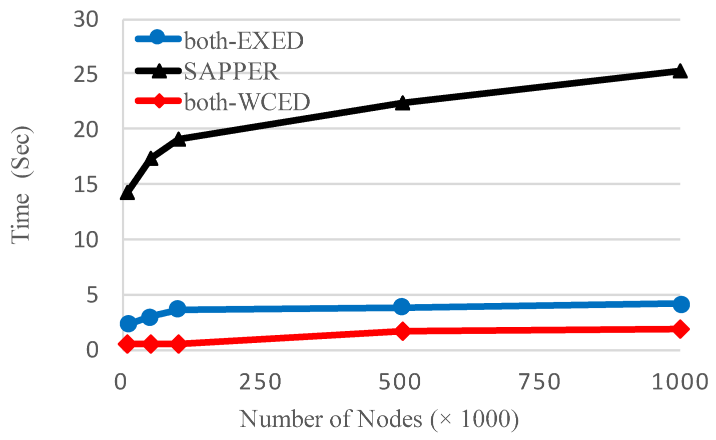


Figure 5.11: Time vs different algorithms

In the second experiment, we use the same setting as the first one, except that we do *not* limit the number of answers; instead we set a 1500 seconds time limit on each algorithm. The running times for SAPPER and our algorithms are reported in Figure 5.12. The graph shows that SAPPER is not a viable solution to our problem for data graphs of realistic sizes: the running time of SAPPER grows much faster than our algorithms, and quickly hits the 1500 seconds time limit in the data graph with 100K nodes. WCED with both filtering schemes exhibits the best performance. We also observe that the time cost of our algorithms scales gracefully with the number of nodes in the data graph. This meets our expectation, because the number of relevant answers increases rapidly (See Figure 5.13) as the number of data nodes increases.

Comparing against Exemplar queries: To evaluate the effectiveness of our queries, since exemplar queries[23] has proved that their queries outperform NeMa[18], we only compare our queries to the exemplar queries. We chose 10 queries (See the Appendix) from the AOL query log and manually mapped them to subgraphs in freebase. For each query, we introduce errors

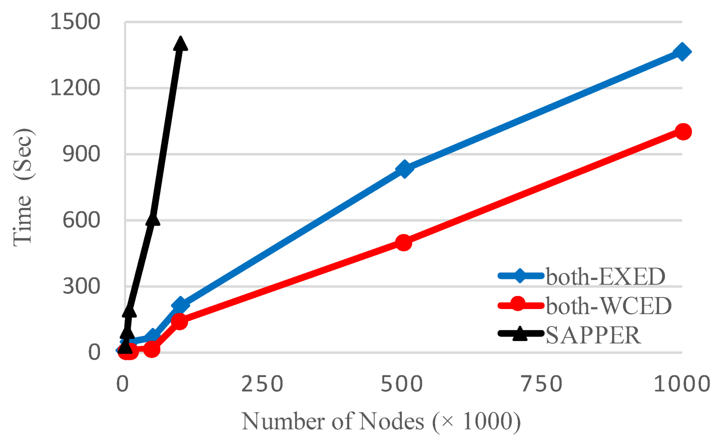


Figure 5.12: Time vs different algorithms

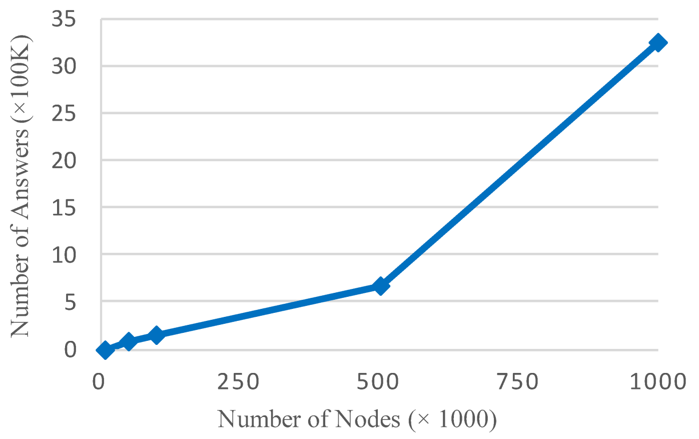


Figure 5.13: Number of answers vs number of nodes in data graph

by randomly selecting an edge and replacing its label with a label randomly selected from the data graph. The number of query edges ranged between 6 and 8. To control the size of the answer set (and to avoid a blow-up), we varied the edit distance threshold from 0 to 2. When the edit distance threshold is 0, our queries are identical to exemplar queries. As expected and shown in Figure 5.14, exemplar queries fail to return any answer for queries with errors and the larger the edit distance thresholds are, the more answers are returned.

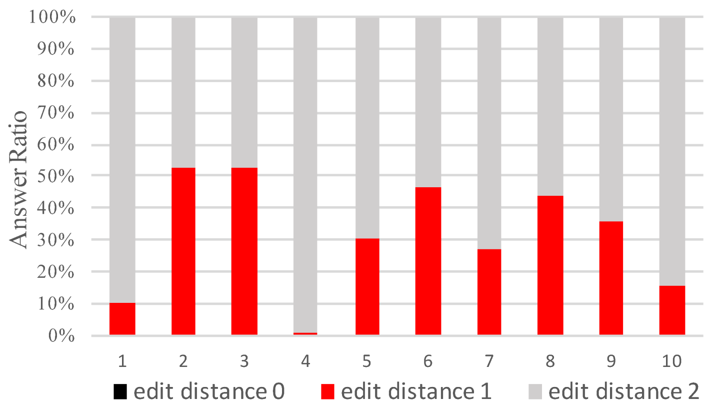


Figure 5.14: Answer set composition

To evaluate the quality of ETEQ answers, we conducted the following user study. We asked 10 users (students in University of Alberta) to evaluate our system. For each query in the test set, we provided an explanation of the topic, the query intention, and our answer set with different edit distances. We asked each user to rate each result as irrelevant, weakly related, or very related with respect to the topic and the expressed query intent. Due to the large size of the answer sets, for each answer set and each edit distance, we randomly chose up to 10 answers for evaluation. We observe in Figure 5.15 that the relevant set has many answers with edit distances 1 and 2. These answers cannot be returned by exemplar queries.

We are not comparing the efficiency of our algorithms against Exemplar because (1) Exemplar does not support edit distance thresholds larger than zero, and (2) our EXED algorithm becomes identical to Exemplar when the

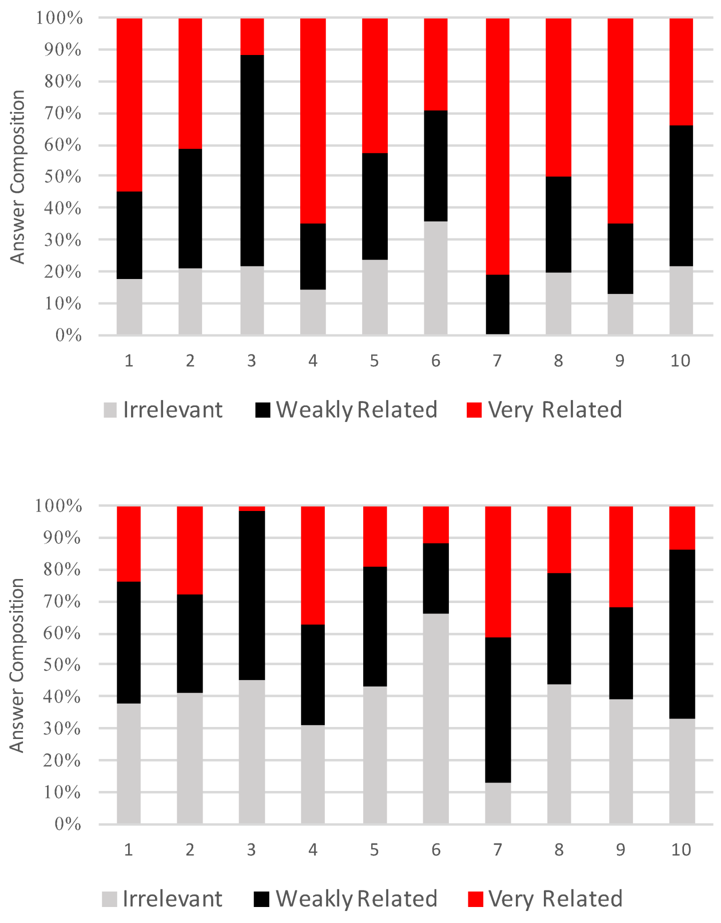


Figure 5.15: Relevant answer set composition for edit distance 1 **(top) and** 2 **(bottom)**

edit distance threshold is zero and we have extensively evaluated EXED with different edit distance thresholds.

Chapter 6

Conclusion

In this thesis, we study the problem of error-tolerant exemplar queries on knowledge graphs. Unlike exemplar queries that support only exact matching of the labels, our developed algorithms allow errors in query and data graph. Two filtering techniques (neighbourhood and path filtering) and two algorithms (EXED and WCED) are developed to handle edit operations as well as to facilitate the searching process. Through a comprehensive experimental evaluation on real and synthetic datasets, we show that our algorithms are both efficient and effective, outperforming existing algorithms. As a future work, instead of retrieving all answers for a query, we plan to efficiently retrieve Top-k relevant answers for the query.

Appendices

Query Set

Query ID	Subject	Predicate	Object
1	D	influenced	Swift
	Scala	influenced	Swift
	Ruby	influenced	Swift
	Rust	influenced	Swift
	Swift	languages	Function programming
	Swift	languages	Procedural programming
	Swift	languages	Generic programming
	Swift	subject_of	Treehouse
2	Lloyd Wright	structures_designed	Oasis Hotel
	Lloyd Wright	place_of_death	Santa Monica
	Lloyd Wright	architectural_style	Modern architecture
	Lloyd Wright	influenced_by	Frederick Law Olmsted
	Lloyd Wright	influenced_by	Frank Lloyd Wright
3	National Audubon Society	notable_types	Nonprofit organization
	National Audubon Society	program_partnership_s	Appalachian Mountains Joint Venture
	National Audubon Society	program_partnership_s	Virginia Bird Conservation Initiative
	National Audubon Society	named_after	John James Audubo
4	Pythagoras	namesakes	Pythagoras
	Pythagoras	influenced	Nikohl Vandel
	Pythagoras	children	Damo
	Pythagoras	influenced	Plato
	Pythagoras	influenced	Jbir ibn Hayyn
5	Frederick County	contains	Ole Orchard Estates
	Frederick County	events	Second Battle of Winchester
	Frederick County	partially_contains North Mountain	
	Frederick County	contains	Echo Village
	Frederick County	people_born_here	James Brenton (17401782)
	Frederick County	contains	Green Acres
	Frederick County	contains	US Census 2000 Tract 51069050100

Table 6.1: Query 1 - 5

Query ID	Subject	Predicate	Object
6	Research	subject_of	Carnegie Moscow Center
	Research	works	Hot talk, cold science
	Research	works	Person or Persons Unknown
	Research	organizations_of_this_type	Stanford University School of Medicine
	Research	schools_of_this_kind	Indian Institute of Forest Management
	Research	organizations_of_this_type	Stanford Radiology
7	Valve Corporation	games_developed	Half-Life 2
	Valve Corporation	games_published	Wolfenstein 3D
	Valve Corporation	games_published	The Maw
	Valve Corporation	games_developed	CS Online
	Valve Corporation	games_published	Half-Life 2
	Valve Corporation	is_reviewed	Place founded
	Valve Corporation	games_published	CS Online
8	Scheme	influenced	Haskell
	Scheme	influenced	Clojure
	Scheme	influenced	LFE
	Scheme	influenced	Dylan
	Scheme	influenced_by	Lisp
	Scheme	parent_language	Lisp
9	Cruze Control	genre	Southern hip hop
	Cruze Control	notable_types	Musical Album
	Southern hip hop	artists	Triple C's
	Cruze Control	featured_artists	Baby
	Cruze Control	featured_artists	Pitbull
	Southern hip hop	albums	Cruze Control
10	Luke Carroll	place_of_birth	Sydney
	Marcus Einfeld	place_of_birth	Sydney
	Martin Lynes	place_of_birth	Sydney
	George Tarr	place_of_birth	Sydney
	Adventures by Disney Australia Vacation	travel_destinations	Sydney
	Dayne Hudson	place_of_birth	Sydney

Table 6.2: Query 6 - 10

Bibliography

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[2] Alexandru T Balaban. Applications of graph theory in chemistry. *Journal of Chemical Information and Computer Sciences*, 25(3):334–343, 1985.

[3] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. of the SIGMOD Conf.*, pages 1247–1250, 2008.

[5] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3):255–259, 1998.

[6] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *Proc. of the SODA Conf.*, pages 30–39, 2004.

[7] James Cheng, Yiping Ke, Wilfred Ng, and An Lu. Fg-index: towards verification-free query processing on graph databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 857–872. ACM, 2007.

[8] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

[9] Banu Dost, Tomer Shlomi, Nitin Gupta, Eytan Ruppin, Vineet Bafna, and Roded Sharan. Qnet: a tool for querying protein interaction networks. *Journal of Computational Biology*, 15(7):913–925, 2008.

[10] MS Fabian, K Gjergji, and W Gerhard. Yago: A core of semantic knowledge unifying wordnet and wikipedia. In *16th International World Wide Web Conference, WWW*, pages 697–706, 2007.

[11] Mirtha-Lina Fernández and Gabriel Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6):753–758, 2001.

[12] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

[13] Rosalba Giugno and Dennis Shasha. Graphgrep: A fast and universal method for querying graphs. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 112–115. IEEE, 2002.

[14] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[15] Nandish Jayaram, Arijit Khan, Chengkai Li, Xifeng Yan, and Ramez Elmasri. Querying knowledge graphs by example entity tuples. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2797–2811, 2015.

[16] Haoliang Jiang, Haixun Wang, S Yu Philip, and Shuigeng Zhou. Gstring: A novel approach for efficient search in graph databases. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 566–575. IEEE, 2007.

[17] Arijit Khan, Nan Li, Xifeng Yan, Ziyu Guan, Supriyo Chakraborty, and Shu Tao. Neighborhood based fast graph search in large networks. In *Proc. SIGMOD Conf.*, pages 901–912, 2011.

[18] Arijit Khan, Yinghui Wu, Charu C Aggarwal, and Xifeng Yan. Nema: Fast graph search with label similarity. In *Proceedings of the VLDB Endowment*, volume 6, pages 181–192. VLDB Endowment, 2013.

[19] Graham Klyne and Jeremy J Carroll. Resource description framework (rdf): Concepts and abstract syntax. W3C, 2006.

[20] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[21] Anna Lubiw. Some np-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10(1):11–21, 1981.

[22] Misael Mongiovi, Raffaele Di Natale, Rosalba Giugno, Alfredo Pulvirenti, Alfredo Ferro, and Roded Sharan. Sigma: a set-cover-based inexact graph matching algorithm. *Journal of bioinformatics and computational biology*, 8(02):199–218, 2010.

[23] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. Exemplar queries: Give me an example of what you need. *PVLDB*, 7(5):365–376, 2014.

[24] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[25] Michel Neuhaus and Horst Bunke. An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 180–189. Springer, 2004.

[26] Michel Neuhaus and Horst Bunke. Edit distance-based kernel functions for structural pattern classification. *Pattern Recognition*, 39(10):1852–1863, 2006.

[27] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 163–172. Springer, 2006.

[28] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H Pham, Jafar M Al-Kofahi, and Tien N Nguyen. Graph-based mining of multiple object usage patterns. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 383–392. ACM, 2009.

[29] John W Raymond, Eleanor J Gardiner, and Peter Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6):631–644, 2002.

[30] Kaspar Riesen, Stefan Fankhauser, and Horst Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In *MLG*, 2007.

[31] Antonio Robles-Kelly and Edwin R Hancock. Graph edit distance from spectral seriation. *IEEE transactions on pattern analysis and machine intelligence*, 27(3):365–378, 2005.

[32] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment*, 1(1):364–375, 2008.

[33] Ellen Spertus, Mehran Sahami, and Orkut Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. In *Proc. of the KDD Conf.*, pages 678–684, 2005.

[34] Yuanyuan Tian and Jignesh M Patel. Tale: A tool for approximate large graph matching. In *Proc. of the ICDE Conf.*, pages 963–972, 2008.

[35] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[36] Guoren Wang, Bin Wang, Xiaochun Yang, and Ge Yu. Efficiently indexing large sparse graphs for similarity search. *TKDE*, 24(3):440–451, 2012.

[37] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q Zhu. Probase: a probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 481–492. ACM, 2012.

[38] Mohamed Yahya, Denilson Barbosa, Klaus Berberich, Qiuyue Wang, and Gerhard Weikum. Relationship queries on extended knowledge graphs. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 605–614. ACM, 2016.

[39] Xifeng Yan, Philip S Yu, and Jiawei Han. Graph indexing: a frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 335–346. ACM, 2004.

[40] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: on approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

[41] Shijie Zhang, Meng Hu, and Jiong Yang. Treepi: A novel graph indexing method. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 966–975. IEEE, 2007.

[42] Shijie Zhang, Shirong Li, and Jiong Yang. Gaddi: distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 192–203. ACM, 2009.

[43] Shijie Zhang, Jiong Yang, and Wei Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3(1-2):1185–1194, 2010.