Challenges of Navigational Queries: Finding Best Paths in Graphs

Louiqa Raschid[1], María-Esther Vidal[2], Yao Wu[1], Marelis Cardenas[2], and Natalia Marquez[2]

[1] University of Maryland
{louiqa,yaowu}@umiacs.umd.edu
[2] Universidad Simón Bolívar
{mvidal,mcardenas,nmarquez}@ldc.usb.ve

Abstract. Life science sources are characterized by a complex graph of overlapping sources, and multiple alternate links between sources. A (navigational) query may be answered by traversing multiple alternate paths between an origin and target source. Paths may be characterized by several metrics, including the cardinality of objects of the target source(TOC), the cost of query evaluation of a plan for the path, and the user's preference for specific paths. Our challenge is finding the best paths among the set of all solutions, AllPaths, that meet some user specified ranking criteria. If the user ranking criteria is strict, then the problem is to find the Top K paths. If the user wants a trade-off of several metrics, then the problem is to find the Skyline paths that are not dominated by other paths. *NSearch* is a naive solution. *BFSrchOpt* is a heuristic best-first search strategy. It uses a metric to rank partial solutions (subpaths) and (local) metrics to guide graph traversal, and produces BFPaths. We compare the precision and recall of BFPaths compared to the Top K% or Skyline of AllPaths. We study the impact of graph properties on the behavior of *BFSrchOpt*. *BFSrchOpt* can be orders of magnitude faster than *NSearch*.

1 Introduction

During the past few years, the number of biomolecular Web accessible sources has increased rapidly. For a particular molecular concept, e.g., gene or protein, there may be several sources, each of which may have several links to other Web sources. To integrate data across these sources, users traverse alternate links and paths through sources. Given a navigational query, the space of possible paths can be exponential in the number of sources and links that are relevant to the query [11]. Further, once paths have been identified, the user still has to explore all the results in the target sources, e.g., all the publications linked to some protein. Since this is a time intensive exercise, it is important to support a user and try to identify the best paths to answer a navigational query.

Suppose we consider a specific path. The number of target objects in the target source reached along the path is one metric characterizing the path. Scientists also have their own preferences for a specific source or link. For example, if a user wants information on proteins he may specify a preference for SwissProt over PIR. User preference is another metric for this path. Since data is distributed across remote sources and there are delays, or charges for accessing servers, the evaluation may vary among the different paths. The diversity of metrics that describe a path results in the user having to solve a difficult task of selecting the best through life science sources. When the search space is large, then this is both, difficult and expensive.

The challenge that we address in this paper is quickly finding the paths that can provide answers to a navigational query, and then finding the best paths from among the set of all paths AllPaths. If the user has a strict ranking criterion based on a specific metric for the paths, then the problem can be modeled as a Top K search. However, if the user is interested in multiple metrics that are not comparable, e.g., cost and user preference, then the problem is to quickly find the Skyline paths that are not dominated by other paths in AllPaths, for these metrics.

We consider the problem of a navigational query which is expressed as an extension to a regular expression. A query is answered by a set of paths from an origin source to a target source. The user also specifies a metric to solve the Top K problem, or a tradeoff of several metrics to solve a Skyline problem.

We analytically determine an upper bound on the number of paths for a variety of graphs and demonstrate that the search space of paths cannot be efficiently explored by an exhaustive algorithm. For comparison purposes we use a naive search *NSearch* [10] based on a Deterministic Finite Automaton (DFA) for a query. It exhaustively enumerates the space of the paths for the query and tries to produce all solutions in AllPaths.

This paper presents the *BFSrchOpt* algorithm that implements a first-best strategy to traverse the space of sources and links, guided by the DFA of the query. It uses a heuristic designed to rank and minimize the number of possibilities considered at each point. Thus, it reduces the time needed to produce good paths. For the Top K problem with a strict ranking on a metric, *BFSrchOpt* will output the first K paths traversed. For the tradeoff of several metrics and the Skyline problem, *BFSrchOpt* will identify a set of non-dominated source paths. We use a divide and conquer based algorithm presented in [8], to identify the set of non-dominated paths.

We use three metrics: target object cardinality (TOC), evaluation cost, and user preference (UP). We consider the Top K problem for each metric and the Skyline problem for the tradeoff of the three metrics. We present the results of an experimental evaluation of *BFSrchOpt* for a diversity of synthetic graphs. We also consider pseudo-real graphs that reflect the metrics of some real sources at NCBI, NIH. We compare the behavior of the Top K or Skyline of paths produced by *BFSrchOpt* with respect to the Top K or Skyline of AllPaths, if we can generate them. We report on the number of paths produced by the *BFSrchOpt* when the different metrics are considered. We use precision and recall for this comparison. We study the effect of the shapes of the graphs, the size of the graphs, and metrics describing the links. The precision and recall of *BFSrchOpt* are both high across all graphs and indicate uniformly good behavior across a wide range of graphs and metrics. *BFSrchOpt* outperformed the computation time of *NSearch*, often by multiple orders of magnitude.

The paper is organized as follows: in Section 2 we motivate the problem using an example. Section 3 describes our approach. Section 4 describes a naive solution and presents *BFSrchOpt*. Section 5 presents the results of an experimental study. Section 6 compares our proposed approach with respect to related work. Finally, in section 7, we give our conclusions.

2 Motivating Example

In Figure 1, a subset of sources at the National Center for Biotechnology Information (NCBI)(http://www.ncbi.nlm.nih.gov) is presented. Sources are associated with logical concepts, Gene, Protein, Sequence, Publication. Each source has some objects stored within, each having zero or more links to objects in the other sources.

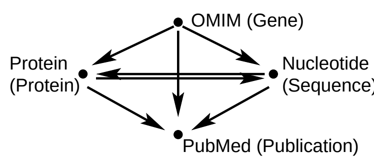

Fig. 1. A source graph for OMIM and NCBI data sources (and corresponding scientific entities)

Path	TOC	Cost	UP
P_1:OMIM → PubMed	4112	4584	0.75
P_2:OMIM → Nuc → PubMed	5891	3512	0.9
P_3:OMIM → Protein → PubMed	6612	3851	0.9
P_4:OMIM → Nuc → Protein → PubMed	5847	5000	0.50
P_5:OMIM → Protein → Nuc → PubMed	5800	12379	0.7

Table 1. Five paths from OMIM to PubMed

Suppose a scientist needs to interrogate these sources, and retrieve all publications that are linked to gene instances. To answer this query, a set of source paths in Figure 1 from OMIM to PubMed will be explored. Table 1 lists the five source paths connecting the origin source OMIM and the target source PubMed. Each path is described by the number of publication entries that are reached following the path or cardinality of target objects (TOC), the cost in milliseconds for retrieving these objects from the sources (Cost) [3], and the user preference (UP). These metrics will be discussed in detail later.

Even with this small number of sources in Figure 1 there are 5 paths. As the number of paths in the solution increases, the user may be interested in a recommendation of the best paths, based on their chosen metrics. Suppose the user wanted to see the top K=3 paths and the metric was user preference UP. Then, the best 3 paths are P_2, P_3 and P_1, where P_2 and P_3 have the highest value.

On the other hand, a user may consider all three metrics to be equally important. In this case the two solutions P_2 and P_3 are identical on UP, but they are incomparable with respect to TOC and Cost since the TOC for P_3 is greater than P_2, and P_3 has higher cost. Then, P_2 and P_3 are the non-dominated paths. They will be in the Skyline.

3 Querying Life Science Sources

In this section, we introduce our life science data model and query language. Second, we present a theoretical analysis of the size of the resulting space of source paths. Third, we define the Top K and Skyline problems. Finally, we describe three metrics that can characterize a source path.

3.1 Data Model

Life science sources may be modeled at three levels: the physical level, the object level and the logical level. The logical level is represented by a directed *Logical Graph*, where nodes correspond to logical classes, C, e.g., protein, gene, publication, and edges represent relationships between logical classes. The physical level corresponds to the actual data sources S and the links L_S that exist between them. An example of data sources and links is shown in Figure 1. The physical level is modeled by a directed *Source Graph*. An *Object Graph* is a graph (O, L_O), where O is a set of objects and L_O is a set of links between objects. ϕ maps a logical class in C U $\{\epsilon_C\}$ (a wildcard) to the set of sources in 2^S that implement it; the wildcard ϵ_C is mapped to all the sources in S. m_O is a mapping from each object in O to some source S.

3.2 Query Language and Semantics

The query language is an (extended) regular expression over the alphabet of C, and ϵ_C represents any logical class. The BNF is in Figure 2. The result of the evaluation of a query Q is two sets of simple paths S_p and S_o. Each path in S_p corresponds to a path in the *Source Graph* that implements Q. Each path in S_o is a path in the *Object Graph* that implement some path in S_p. Details in [9, 10].

[3] Numbers presented in Table 1 correspond to a sample of the data from the NCBI sources.

Regular expressions express navigational queries. For example, a scientist may be interested to *Retrieve all publications (p) linked to proteins (r)*; this is expressed as $(r.p)$. Class r can be interpreted by either NCBI Protein or Swiss-Prot. Class p can be interpreted by PubMed. Suppose that there is a link from NCBI Protein to PubMed, and a link from Swiss-Prot to PubMed. Therefore both links, (Protein → PubMed) and (Swiss-Prot → PubMed), are paths in S_p that implement $(r.p)$.

Consider a query *Retrieve publications linked to genes via any number of intermediate sources (paths of any length* ≥ 2*)*; it is (g.ϵ^+.p). If we answer this query using the *SourceGraph* of Figure 1, there will be five paths from OMIM to PubMed in the solution S_p. We note that in general a user will be interested in applying selection criteria to identify specific proteins or genes or publications in specific sources. For simplicity we do not consider such extensions in this paper.

Finally, the query language allows the user to select either the Top K option (value K and specific metric) or the Skyline option (the set of metrics to be considered for the tradeoff). The metrics are TOC, UP and Cost, and will be described later.

Query := Query Ranking
| “(” Query “)”| “(“Query”)∗”
| Query “|” Query | Query . Query
| ϵ_C | ClassName
Ranking := empty | TopK | Skyline
TopK := Top K Metric
K := integer
Skyline := Skyline tradeOff Metrics
Metrics := Metric “,” Metric | Metric “,” Metrics
Metric := “min” | “max” MetricName
MetricName:=“TOC”| “UP” | “Cost”

Fig. 2. The Syntax of the Query Language

3.3 The Size of the Search Space

To motivate the importance of the problem of efficiently traversing the search space, we discuss the size of the search space. We consider a number of graphs, ranging from a binary tree, a chain tree to a DAG. All graphs considered are acyclic, otherwise we would have an infinite search space. Let n be the number of nodes in the graph (tree); this is the number of nodes in the *Source Graph*.

Let l be the length of the regular expression. We consider when the regular expression (RE) allows (does not allow) the * to represent paths of unspecified

SG and RE	binary complete tree	chain tree	general tree	DAG
ε free RE-allow *	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(\binom{n}{2} \cdot \frac{1}{n}\binom{2(n-1)}{n-1})$
* free RE-allow ε	$O(l \cdot n)$	$O(n-l+1)$	$O(l \cdot n)$	$O(\binom{n}{l} \cdot (l!))$
general RE-allow * and ε	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(\binom{n}{2} \cdot \frac{1}{n}\binom{2(n-1)}{n-1})$

Table 2. Search space for different RE and SG

length. We also consider when RE allows (does not allow) the wild card ϵ. Recall that a wildcard would match against any of the class labels of some set E.

Table 2 reports on the upper bound of the number of paths in the search space corresponding to different classes of RE and for different types of the SG. We note that the highest cardinality is for REs that allow * for the DAG.

If there is a * in RE, then all paths in the source graph could satisfy RE. In a tree type source graph, there is exactly one path between each pair of source nodes. For a DAG type graph, the number of paths between any pair becomes *Catalan number* series, and there are at most $\binom{n}{2}$ pairs. If there is no * in RE, then the number of paths is a function of the length of the RE.

3.4 Two Best Path Problems

Top K Given a query Q, a Source Graph SG, a metric M and an integer K, the *Top K* problem is identifying the K paths S'_p=$\{p_1,\ldots,p_k\}$, with the highest ranking for M.

For example, a user may be interested only on the top 3 paths that minimize the cost of retrieving publications reached from genes via any number of intermedia sources; using our language this is specified as follows:

(g.ϵ^+.c Top 3 min Cost).

If we evaluate the query against source in Figure 1, then the result will be the paths P_1, P_2 and P_3 from Table 1.

Skyline Given a query Q, a Source Graph SG, and the tradeoff of a set of metrics SM all equally important for the ranking, the *Skyline* problem is identifying the non-dominated set of source paths S'_p=$\{p_1,\ldots,p_j\}$, where paths in S'_p have the highest ranking for the metrics SM and all paths are incomparable for the metrics.

Suppose a user is interested in paths that maximize the numbers of publications reached from genes via any number of intermediate sources and that minimize the cost. The query is as follows: (g.ϵ^+.c Skyline tradeOff max TOC, min Cost).

Considering the source in Figure 1, we will find out that source paths P_3 and P_2 are better than any other path, but that they are incomparable, i.e., none of them is better than the other in terms of TOC and Cost. Then, both P_3 and P_2 are the top paths and will comprise the answer to the Skyline Problem.

A set of non-dominated paths consists of all paths that are not dominated in all the metrics by any other path in the sense of Pareto optimality. This set is also called Pareto set or Skyline [5, 13]. The problem of finding a set of non-dominated solutions is referred in the literature as the maximal vector problem [8, 5]. In [8] a theoretical algorithm based on a divide and conquer strategy was presented. The algorithm established that the maximal vector problem is $o(n^2)$, where n is the size of the set of vectors. Fortunately, the divide and conquer is not as expensive and its average cost is $O(mnlogn)$, where m is the number of dimensions of the vectors, and n is the size of the set of vectors.

3.5 Source Path Metrics

A source path p in a *Source Graph* can be characterized by a number of metrics. In this paper, we consider three metrics, namely, the cardinality of the target objects (TOC), the cost of evaluating a path (Cost), and a metric to represent user's preferences (UP).

Target Object Cardinality The *Target Object Cardinality (TOC)* is defined as the number of distinct objects that are reached in the target (final) source of the path [18]. TOC can be computed by evaluating a set of join operations or by sampling all the objects paths into the Object Graph. This can be quite expensive and here we present an estimation of TOC. Consider a path p through sources $S_1, S_2, \ldots, S_n$, TOC for p is defined as follows:

$$TOC(p) = c(S_n)tocf(n) \tag{1}$$

The $tocf(i)$ is a factor representing the probability that an object in S_i can be reached from some object in S_1. The value of $tocf(i)$ for a path from S_1 to S_2 is trivially defined as $tocf(2) = l_{im}(S_{1,2})/c(S_2)$, where $l_{im}(Si, i+1)$ corresponds to the number of data objects in S_{i+1} that have at least one incoming link from objects in S_i, and $c(S_{i+1})$ represents the cardinality of source S_{i+1}.

To compute $tocf(i+1)$ we compute the probability that an object is not reached for some $tocf(i)$, and then we use that to compute the probability that it is reached. An object x in S_{i+1} receives on average $\delta^{in}_{i+1} = l(S_{i,i+1})/l_{im}(S_{i,i+1})$ edges from objects in S_i. If at least one of these δ^{in}_{i+1} objects in S_i is reached from some object in S_1, then we will reach x. Similarly, an object x will not be reached if all δ^{in}_{i+1} objects in S_i are not reached from some object in S_1. The expression $tocf(i+1)$ is recursively defined in terms of $tocf(i)$ as follows:

$$tocf(i+1) = (l_{im}(S_{i,i+1})/c(S_{i+1})(1 - (1 - tocf(i))^{\delta^{in}_{i+1}}) \tag{2}$$

where, $\delta^{in}_{i+1} = l(S_{i,i+1})/l_{im}(S_{i,i+1})$, and $l(S_{i,i+1})$ is the number of links from all data objects of source S_i pointing to data objects of S_{i+1}.

User Preference The user preference metric defines the preference of a user for a path in terms of her preferences for the sources and links in the path. We assume users provide an absolute score between 0.0 and 1.0 to measure their preferences for all the sources implementing class C or all links between sources that implement a logical link between classes C_i and C_j. Table 3 shows a simple example. Note that we expect that users will provide such rankings specific to certain queries, e.g., if the user is interested in some subset of proteins that are associated with "apoptosis", then she will express a preference for that particular subset. We expect that user preferences will be updated as queries are evaluated and users examine the results obtained.

Logical	Physical	UP
Protein	SwissProt	0.8
Protein	PIR	0.6
Protein	PDB	0.75
Gene → Protein	OMIM → SwissProt	0.8
Gene → Protein	OMIM → PIR	0.9
Gene → Protein	OMIM → PDB	0.5

Table 3. User Preferences

Let $UP_s(S_i)$ be the user preference for a source S_i and let $UP_l(S_i, S_{i+1})$ be the user preference for a link; both are a value in the range [0.0 - 1.0]. Then, the user preference for a path is $\sum_{i=1}^{n-1} 1/n \times (min(UP_s(S_i), UP_l(S_i, S_{i+1}), UP_s(S_{i+1}))$

Cost of Evaluating a Plan for a Path This metric *estimates* the cost of evaluating a path p in terms of the intermediate objects generated during the evaluation of the path p [18]. We consider that each link from S_j to S_{j+1} in a path p, can be either implemented as a $hashJoin(S_{j,j+1})$ or a *navJoin*($S_{j,j+1}$), and we measure the cost of the path, considering that one implementation of the links is given. Then, cost of path corresponds to the sum of the costs of the evaluation of each link in the query.

A $hashJoin(S_{j,j+1})$ requires a hash table to distinguish the objects that satisfy the join condition, and the cost of $hashJoin(S_{j,j+1})$ is defined as follows [14]:
$cost(hashJoin(S_{j,j+1})) = 3(c(S_j) + c(S_{j+1})$

The cost of $hashJoin(S_{j,j+1})$ and $hashJoin(S_{j+1,j})$ is the same. On the other hand, a navigational join (*navJoin*($S_{j,j+1}$)) traverses all the objects in S_{j+1} that are linked to an object in S_j. The cost of a navigational join is similar to the cost of a nested loop join [14], and it is defined as follows: $cost(navJoin(S_{j,j+1}) = c(S_j) + c(S_j) \times c(S_{j+1})$. The cost of *navJoin*($S_{j,j+1}$) and *navJoin*($S_{j+1,j}$) may not be the same since the number of objects in S_j and S_{j+1} may be different.

4 Best Paths Algorithms

In this section we describe our algorithm in detail. We begin describing a naive approach, second, we present some basic assumptions, and finally, we propose the *BFSrchOpt* algorithm.

4.1 Naive Solution

A naive solution for the problem of producing the Top K source paths, can be performed in three steps: a) produce all the source paths that implement a query using *NSearch*, b) probe the ranking function for each path, and c) output the K paths with the highest scores. This can be done as described in [17]. If the trade-off of the metrics is considered, then the set of non-dominated paths should be returned. This naive approach requires an exhaustive search of the source paths in the *Source Graph*, a sequential scan of the complete set of paths to compute the metric values, and the ranking of the complete set of paths. As we showed in the previous section, the number of source paths can be exponential on the size of the graph, and a complete probing at query time is clearly unacceptable in most cases.

In previous research in [10] we presented the algorithm *NSearch* that is based on a deterministic finite state automaton (DFA) that recognizes a regular expression. The *NSearch* algorithm runs in polynomial time in the size of the graph, if the graph is cycle-free and all paths are cycle-free. If d is the maximum number of sources that can precede a source in the SG, and b is the maximum length of (cycle free) paths satisfying the regular expression, then $O(d^b)$ is an upper bound for *NSearch*.

4.2 A Best First Strategy

The naive solution presented in the previous subsection, requires to explore the complete space of paths to produce the first solution. In large spaces, this behavior is not acceptable. To address this problem, we propose the *BFSrchOpt* algorithm that implements a best first strategy that produces top source paths relatively fast. Figure 3 shows algorithm *BFSrchOpt*.

Basic Principles *BFSrchOpt* traverses the space of paths using a DFA that represent the regular expression or query. At each step of the search, *BFSrchOpt* ranks and chooses the K best subpaths (Top K Problem) or the set of non-dominated subpaths (Skyline Problem). It expands these subpaths using sources that are ranked using local metrics. The divide and conquer based algorithm proposed in [8], is used at each step to identify the set of non-dominated subpaths. The *BFSrchOpt* uses the following four local metrics. Each local metric was defined to maximize/minimize the values of its corresponding metric, for example, *ltoc*(S_j) allows to rank the sources that maximize the TOC, and so on.

- Top K for TOC: *ltoc*(S_j)=$((1 - tocf(j-1))^{\delta^{in}_{S_j}})$.
- Top K fro Cost: *lcost*(S_j)= the minimum of $hashJoin(S_{j-1,j})$, $navJoin(S_{j-1,j})$, and $navJoin(S_{j,j-1})$.
- Top K for UP: *lup*(S_j)= the minimum of $UP_s(S_j)$ and $UP_l(S_{j-1}, S_j)$.
- Skyline for TOC-Cost-UP: *ltoc-lcost-lup*: maximizes *ltoc*(S_j) and *lup*(S_j) and minimizes *lcost*(S_j)

To avoid considering only locally optimal subpaths, *BFSrchOpt* chooses also a number of subpaths that decrease the metric measure. Thus, if p is the subpath, identified for the current transition t, that has the greatest metric value, then, *BFSrchOpt* will choose subpaths p_i, different from p, that decrease the metric value. The probability of choosing p_i is given by a Boltzmann factor: $e^{-((f(p_i)-f(p))/T)}$, where f denotes the metric and T is a function that represents how far the transition t is from a final state in the DFA. The value T decreases as the *BFSrchOpt* gets closer to final states in the DFA, and the probability of choosing a majority of good solutions gradually increases. This technique to choose subpaths is similar to the one used in the meta-heuristic Simulated Annealing to avoid becoming trapped in a local minima.

Algorithm BFSrchOpt
INPUT: DFA: A deterministic finite state automaton for the query. SG: Source Graph. ϕ: Mapping from logical concepts to set of sources. K: The number of subpaths considered in each iteration. SM: Metrics. TradeOff: Boolean. LM: Local Metrics.
OUTPUT: FINAL: a set of Top paths.

1. **INITIALIZE**:
   (a) Assign Empty to FINAL.
   (b) Create stack OPEN of subpaths, where a subpath s is a list of pairs, (S_i, t_i). A t_i is a transition of the DFA and it is a triple (i,f,e), where, i is the initial state, f is the final state, e is a class label in E. Initialize OPEN with a subpath with a singleton pair in the list, corresponding to the first transition of the DFA, where i is the start state of the DFA.
2. **RANK**: If $TradeOff$ identify the non-dominated subpaths in OPEN; Else, use metrics in SM to choose the top K paths in OPEN.
3. **SEARCH**: While OPEN is not empty
   (a) Select the highest ranking subpath of OPEN, s_h=p_1,...,p_l, and consider the last pair p_l=(S_l,t_l) where S_l is a source, and t_l is a transition.
   (b) **EXPAND**:
      i. Create a stack OPENSOURCES for those sources $S\prime \in \phi(e)$, where e is the label in transition t_l and there is a link $(S_l,S\prime)$ in SG.
      ii. Rank OPENSOURCES using the local metrics in LM.
      iii. While OPENSOURCES is not empty
         A. Select the highest ranking source $S\prime$.
         B. If f, the final state of t_l is a final state in the DFA, then push s_h to FINAL. Else, create a pair $(S\prime, t_{next})$, where t_{next} is the transition to follow t_l in the DFA. Append this pair to s_h in OPEN.
         C. If TradeOff or the cardinality of FINAL is K, **EXIT**;
         Else **RANK**.
4. **EXIT** return FINAL;

Fig. 3. Algorithm BFSrchOpt

5 Experiments

We report on an extensive comparison of *NSearch* and *BFSrchOpt* for a variety of graphs where we vary the type of graph (wheel, ring, general) and the size of the graph (cardinality of nodes and edges) and other metrics of the graph, e.g., source and link cardinality, target source image and origin source participation. We include pseudo-real graphs that reflect the metrics of real sources at NCBI.

We generate AllPaths, all solutions to the query, when it is possible, and select the Top 25 % of AllPaths, labeled Top25%Paths, using: TOC, UP and Cost. In the case of the tradeoff, Skyline corresponds to the set of non-dominated solutions in AllPaths. Since the number of paths can be exponential in the size of the graph, *NSearch* may fail to exhaustively enumerate the whole space of paths. We modify the *NSearch* and *BFSrchOpt* algorithms to run for a number of iterations, where an iteration corresponds to a node visited during the search. We run *NSearch* and *BFSrchOpt* for some fixed number I of iterations and generate NPaths and BFPaths. We compare the precision and recall of NPaths and BFPaths with respect to Top25%Paths and Skyline. The precision of BFPaths reflects the efficiency of *BFSrchOpt* in traversing the graph. A high precision or high efficiency indicates that *BFSrchOpt* made good decisions while traversing the graph to generate paths. The recall of BFPaths reflects the performance of *BFSrchOpt* and how close it is to the optimal solution. A high recall or high performance indicates that *BFSrchOpt* made good decisions in solving in the problem of identifying good paths.

We ran all our experiments on a machine equipped with two Pentium III processors running at 950 MHz, 1GB of RAM, Linux Fedora Core Release 1 and Java 1.4. In some cases where the graph was very large, we could not compute AllPaths in a reasonable running time. In such cases, we compare the quality of BFPaths and NPaths and the running time to produce them.

To summarize our results, the precision and recall of *BFSrchOpt* is high across all experiments and indicate uniformly good behavior across a wide range of graphs. The varying *Metrics* and *Local Metrics* all performed well. The metric Cost appeared to have a moderately significant impact in some cases. This may be because it is a monotone function. *BFSrchOpt* outperformed the computation time of *NSearch*, often by multiple orders of magnitude. We now present detailed experimental results.

5.1 Experiment 1: Performance for a Variety of Graph Shapes

We study *BFSrchOpt* and *NSearch* when the shape of the source graph SG changes. Recall that each node in SG is a source containing data objects and that each object has links in the Object Graph OG to objects in other sources. The number of nodes and edges and the shape of SG may impact the performance of both algorithms. Table 4 reports on the number of nodes and edges for each SG, and the minimum, maximum, average and the standard deviation for the indegree and outdegree of the nodes in the synthetic graphs. These graphs were randomly generated following a uniform distribution.

We evaluate the complex query "$a\epsilon^*b$" on three shapes for SG, ring, wheel and general. The total number of paths satisfying the query, AllPaths, for each of the three source graphs is reported in Table 5. We run the algorithms for I iterations and report on the number of paths produced by *BFSrchOpt* and *NSearch*, namely BFPaths and NPaths. We consider three metrics Target Object Cardinality (TOC), Cost, User Preference (UP), to rank BFPaths. *BFSrchOpt* can explore up to K subpaths at each step; we report results for values of K=25%

	Ring	General	Wheel
Nodes	39	84	118
Edges	66	253	226
In-Degree	(1,5,1,1.1)	(0,10,3,2.1)	(1,12,1,2.1)
Out-Degree	(0,6,1,1.3)	(0,21,3,5.1)	(0,14,1,2.8)

Table 4. Parameter settings for graphs of Experiment One

and 100%. We also report on the *Skyline* paths produced by *BFSrchOpt* using the trade-off of these 3 metrics.

	Ring	General	Wheel
Iterations	3000	5000	2000
AllPaths	677	1145	875
NPaths	12	36	31
BFPaths K=100%			
TOC	594	1004	314
Cost	594	1004	306
UP	594	1004	219
BFPaths K=25%			
TOC	170	287	219
Cost	170	287	219
UP	76	286	78
BFPaths Skyline			
TOC-Cost-UP	169	286	218

Table 5. A Comparison of Solutions Allpaths, NPaths and BFPaths for Experiment One

As seen in Table 5, as K increases, *BFSrchOpt* produces more paths. With K=100%, *BFSrchOpt* generates almost all paths in AllPaths for the Ring and General *SG*. It does not perform as well on the Wheel *SG* and produces at most 314 or 40% of AllPaths. We note that in comparison, *NSearch* is unsuccessful in producing many paths, for e.g., it only produces 12 out of 677 possible paths in AllPaths for the Ring *SG*.

	Ring	General	Wheel
Running Time BFSrchOpt K=100%			
TOC	3373	4006	5174
Cost	4518	3696	3277
UP	4477	3603	2972
Running Time BFSrchOpt Skyline			
TOC-Cost-UP	4223	3693	3296
Running Time NSearch			
	1390300	16597990	2663218

Table 6. Running Time (msec) for BFSrchOpt K=100%, BFSrchOpt Skyline and NSearch for Experiment One

Table 6 reports on the time for I iterations for *BFSrchOpt* with K=100%, *BFSrchOpt Skyline* and *NSearch*. As can be seen, *BFSrchOpt* outperforms *NSearch*, and in some cases the performance improvement is several orders of magnitude.

Finally, we compare the precision and recall of BFPaths and NPaths compared to AllPaths. In Figure 4, we show the precision (a) and recall (b) of BFPaths for the 3 metrics TOC, Cost and UP, with K=25% of the size of AllPaths and the Skyline of BFPaths, compared to AllPaths. Our first observation is that the precision and recall values are uniformly good for BFPaths across all graph shapes. The precision varies from between 60 to 100%. Also, Figure 4 reports on the precision (c) and recall (d) of NPaths. The precision is surprisingly good, and in the case of the Wheel SG, it outperforms *BFSrchOpt* for K=25%. However, the recall is uniformly poor compared to BFPaths; in all cases the recall for NPaths ranges from 0 to 15 %.

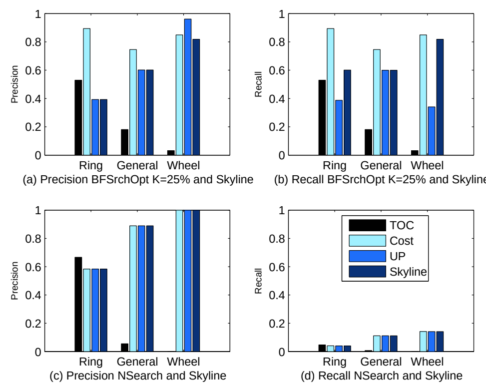

Fig. 4. Precision (a) and Recall (b) of BFSrchOpt K=25% and BFSrchOpt Skyline; and Precision (c) and Recall (d) of NSearch wrt AllPaths and Skyline. Experiment One

To summarize, *BFSrchOpt* has good precision and recall across all graphs and significantly outperforms *NSearch*.

5.2 Experiment 2: Performance for Varying (Pseudo-Real) Graph Sizes

We consider three general graphs G_1, G_2 and G_3 for the source graph SG, of increasing size. They represent pseudo-real graphs. The graph G_1 was constructed to reflect the object graph metadata of NCBI sources that implement the logical concepts gene, nucleotide, citation and protein. Then, a graph $Gi + 1$ was generated from G_i by adding nodes and edges, while maintaining the same metadata as G_i. Table 7 reports on the number of nodes and edges for each graph and

	G_1	G_2	G_3
Nodes	21	40	80
Edges	79	140	280
In-Degree	(0,14,3,3.8)	(0,20,3,4.11)	(0,30,3,4.87)
Out-Degree	(0,8,3,2.5)	(0,15,3,3.65)	(0,23,3,4.52)

Table 7. Parameter setting for pseudo-real graphs of Experiment Two

the minimum, maximum, average and the standard deviation for the indegree and the outdegree of objects in the pseudo-real source graphs.

We evaluate the complex query "$g\epsilon^*n$". The algorithms are run for 950 iterations for G_1 and G_2 and for 6000 iterations for G_3 since this is a large graph. Table 8 reports on the cardinality of Allpaths; as can be seen, it is over 8500 for G_3. The cardinality of NPaths and BFPaths for K=25%, K=50% and Skyline are shown. For these graphs, *NSearch* appears to perform much better than reported in the previous experiment. For G_3, *NSearch* produces slightly more paths compared to *BFSrchOpt* with K=50%.

	G_1	G_2	G_3
Iterations	950	950	6000
AllPaths	1144	1655	8503
NPaths	247	328	2873
BFPaths K=50%			
TOC	398	361	2766
Cost	344	247	1575
UP	283	245	1700
BFPaths K=25%			
TOC	286	361	2126
Cost	286	247	1667
UP	217	245	1441
BFPaths Skyline			
TOC-Cost-UP	217	245	1565

Table 8. A Comparison of Solutions AllPaths, NPaths and BFPaths for Experiment Two

Table 9 reports on the running time for these experiments. *BFSrchOpt* outperforms *NSearch* by at least one order of magnitude.

Figure 5 reports on the precision (a) and recall (b) for BFPaths for K=100% and *BFSrchOpt* Skyline, and it also reports on the precision (c) and recall (c) for *NSearch*.

We note recall for BFpaths and NPaths have similar behavior, and that BFPaths outperforms NPaths for precision. We note that the metric Cost appears to lead to excellent behavior of *BFSrchOpt*. Given that the Cost metric is monotone these results indicate that *BFSrchOpt* was able to produce local top subpaths, that conduced to the top paths.

To summarize, for the general graphs of this experiment, BFPaths and NPaths showed similar behavior, with BFPaths outperforming NPaths moderately. We

Running Time BFSrchOpt K=100%			
Metric	G_1	G_2	G_3
TOC	995	1485	64020
Cost	855	1068	42112
UP	707	1101	46051
Running Time BFSrchOpt Skyline			
TOC-Cost-UP	749	1128	45651
Running NSearch			
	G_1	G_2	G_3
	5496	9958	747335

Table 9. Running Time (msec) for BFSrchOpt K=100%, BFSrchOpt Skyline and NSearch for Experiment Two

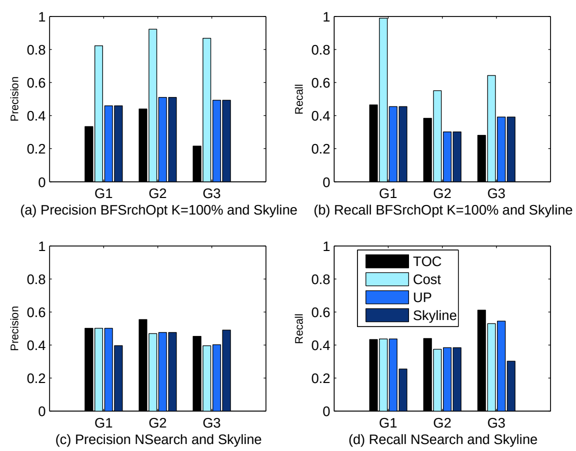

Fig. 5. Precision (a) and Recall (b) of BFPaths K=100% wrt AllPaths; and Precision (c) and Recall (d) of NPaths wrt AllPaths. Experiment Two

observe that *BFSrchOpt* was able to exploit the metric Cost for these graphs. Further, the running time of *BFSrchOpt* is at least one order of magnitude better than *NSearch*.

5.3 Experiment 3: Performance on Very Large Graphs

We compare *BFSrchOpt* and *NSearch* for a very large graph, i.e., it is not feasible to compute AllPaths. Table 10 reports on the graph parameter settings.

	General
Nodes	64
Edges	328
In-Degree	(0,15,5,4.5)
Out-Degree	(0,16,5,4.9)
Iterations	2500
NPaths	298
BFPaths K=100%	
TOC	184
UP	184
Cost	184
BFSrchOpt Skyline	
TOC-Cost-UP	184

Table 10. Parameter settings for Experiment Three

We evaluate the complex query "$h\epsilon^*m$". We ran the experiment for 2500 iterations. We note that *NSearch* generated about a third more paths compared to *BFSrchOpt* for K=100%. However, if we consider the running time reported in Table 11, we observe that the *NSearch* performance is at least 3 orders of magnitude worse compared to *BFSrchOpt*. In other words, *NSearch* performs a significant amount of computation to produce a moderate increase in the cardinality of NPaths. We could not compare precision and recall since we could not generate AllPaths. We did however, compare the TOC, Cost, UP and the trade-off. We have observed that they have similar metrics.

	General
Running Time BFSrchOpt K=100%	
TOC	405
Cost	353
UP	400
Running Time BFSrchOpt Skyline	
TOC-Cost-UP	367
Running Time NSearch	
	376298

Table 11. Running Time (msec) for BFSrchOpt K=100%, BFSrchOpt Skyline, and NSearch for Experiment Three

To summarize, in very large graphs, *BFSrchOpt* has running time at least 3 orders of magnitude better than *NSearch*. *NSearch* does produce a moderate increase of paths compared to BFPaths. However, the paths that are produced seem to have similar quality with respect to the TOC, Cost and UP trade-off.

6 Related Work

The problem of evaluating top K queries has been extensively studied [2, 1, 7]. Additionally, the problem of finding a set of non-dominated solutions is referred in the literature as the maximal vector problem [8] and a variety of algorithms has been proposed [8, 3, 5]. Typically, these two types of approaches share the challenge of identify the top objects among a set, and minimize the number of probes. We also address the problem of producing the top paths, but in our case the objects to be ranked are not given as an input. *BFSrchOpt* has to explore the space of the source paths that implement a query and at the same time, probe which of them could comprise the top final set.

Many strategies or meta-heuristics have been proposed to solve the problem of graph searching [15]. We propose the use of a first best strategy to produce the top source paths relatively fast. However, we plan to implement other strategies as Simulated Annealing or Tabu Search and study their performance and effectiveness with respect to the *BFSrchOpt*.

Finally, multi-objective optimization for database queries has been previously studied [4, 6, 12, 13, 16, 19]. The trade-off of execution cost versus delay in producing the results was studied in [16] in the context of the Mariposa wide area DBMS. More recently the trade-off of execution cost versus coverage was studied in [4, 12, 19] in the context of Internet sources with overlapping coverage. [6] studied the trade-off between the accuracy of results versus the index space needed to provide approximate answers to queries. Our problem is similar to [12, 16, 19], in that they also developed heuristics to rank sources. However, prior research assumed that the set of all sources that could answer the query (with different coverage or delay or cost) was known a priori. The *BFSrchOpt* addresses the problem of traversing the space of paths and identifying the ones that may reach it to the Top source paths.

7 Conclusions and Future Work

We prepared an extensive evaluation of *BFSrchOpt* on a variety of graph shapes and sizes, on very large graphs, and on pseudo-real graphs. In all cases, the precision and recall of *BFSrchOpt* is high to moderate across all experiments and indicate uniformly good behavior across a wide range of graphs. While the four variants of the Metrics all performed well, the Cost metric seemed to have the best behavior. *BFSrchOpt* outperformed the computation time of *NSearch*, often by multiple orders of magnitude.

In future work, we will test *BFSrchOpt* on both artificial worlds and real world data from NCBI. We will extend *BFSrchOpt* to include some randomized strategies in selecting subpaths. This may help it overcome some of the limitations that were observed, where *BFSrchOpt* did not generate all the paths.

References

1. N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accesible databases. *Proceedings of ICDE*, 2002.
2. K. Chen-Chuan and S. Hwang. Minimal probing: Supporting expensive predicates for top-k. *Proceedings of SIGMOD*, 2002.
3. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. *Proceedings of ICDE*, 2003.
4. A. Doan and A. Halevy. Efficiently ordering query plans for data integration. In *Proceedings of the ICDE*, 2002.
5. P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. *Proceedings of Very Large Data Bases, VLDB*, 2005.
6. S. Guha, D. Gunopulos, N. Koudas, D. Srivastava, and M. Vlachos. Efficient approximation of optimization queries under parametric aggregation constraints. In *Proceedings of the VLDB*, pages 778–789, 2003.
7. I. Ilyas, R. Shah, W. Aref, A. Scott, and A. Elmagarmid. Rank-aware query optimization. *Proceedings of SIGMOD*, 2004.
8. H. Kung, F. Luccio, and Preparata. On finding the maxima of a set of vectors. *JACM*, 1975.
9. Z. Lacroix, Parekh K, and M.E. Vidal. Bionavigation: Selecting optimum paths through biological resources to ev aluate ontological navigational queries. *Proceedings of the DILS Conference*, 2005.
10. Z. Lacroix, L. Raschid, and M.E. Vidal. Efficient techniques to explore paths in life science data sources. *Proceedings of the DILS Conference*, 2004.
11. Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 185–193, 1989.
12. Felix Naumann. *Quality-driven Query Answering for Integrated Information Systems*, volume 2261 of *Lecture Notes on Computer Science (LNCS)*. Springer Verlag, Heidelberg, 2002.
13. Christos H. Papadimitriou and Mihalis Yannakakis. Multiobjective query optimization. In *Proceedings of the ACM Symposium on Principles of Database Systems PODS01*, 2001.
14. Ramakrishnan and Gehrke. *Database Management Systems*. Springer Verlag, 2003.
15. S.Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
16. M. Stonebraker, P. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):048–063, 1996.
17. P. Tsaparas. Using non-linear dynamical systems for web searching and ranking. *Proceedings of PODS*, 2004.
18. M. Vidal, L. Raschid, and J. Mestre. Challenges in selecting paths for navigational queries: Trade-off of benefit of path versus cost of plan. *Proceedings of the Workshop on Web and Databases, WebDB*, 2004.
19. R. Yerneni, F. Naumann, and H. Garcia-Molina. Maximizing coverage of mediated web queries. *Stanford University Technical Report, Computer Science Department*, 2000.