

Challenges in Selecting Paths for Navigational Queries: Trade-Off of Benefit of Path versus Cost of Plan

María-Esther Vidal
Universidad Simón Bolívar
Caracas, Venezuela
mvidal@ldc.usb.ve

Louïqa Raschid
University of Maryland
College Park, MD
louïqa@umiacs.umd.edu

Julian Mestre
University of Maryland
College Park, MD
jmestre@cs.umd.edu

ABSTRACT

Life sciences 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 a start source and a target source. Each of these paths may have dissimilar benefit, e.g., the cardinality of result objects that are reached in the target source. Paths may also have dissimilar costs of evaluation, i.e., the execution cost of a query evaluation plan for a path. In prior research, we developed *ESearch*, an algorithm based on a Deterministic Finite Automaton (DFA), which exhaustively enumerates all paths to answer a navigational query. The challenge is to develop heuristics that improve on the exhaustive *ESearch* solution and identify good *utility functions* that can rank the sources, the links between sources, and the sub-paths that are already visited, in order to quickly produce paths that have the highest benefit and the least cost. In this paper, we present a heuristic that uses *local utility functions* to rank sources, using either the benefit attributed to the source, the cost of a plan using the source, or both. The heuristic will limit its search to some *Top XX%* of the ranked sources. To compare *ESearch* and the heuristic, we construct a Pareto surface of all dominant solutions produced by *ESearch*, with respect to benefit and cost. We choose the *Top 25%* of the *ESearch* solutions that are in the Pareto surface. We compare the paths produced by the heuristic to this *Top 25%* of *ESearch* solutions with respect to precision and recall. This motivates the need for further research on developing a more efficient algorithm and better utility functions.

1. INTRODUCTION

Life sciences sources are characterized by a complex graph of overlapping sources, and multiple alternate links between sources. A navigational query can be answered by a choice of alternate paths between a start source and a target source. Consider the query *Return all citations of PubMed that are*

linked to an OMIM entry that is related to some disease or condition. A scientist may choose the OMIM source, which contains information related to human genetic diseases, as a starting point for her exploration and wish to eventually retrieve citations from the PubMed source. Starting with a keyword search on a certain disease, she can explore direct links between genes in OMIM and citations in PubMed. She can also traverse paths that are implemented using additional intermediate sources, e.g., NCBI Protein and NCBI Nucleotide. Figure 1 illustrates a source graph for four data sources.

While the figure presents a small number of sources and links, the number of public life science sources are in the thousands and increasing. A single query or experiment protocol may be interested in exploring tens of protein data sources, and following all the links from these sources. Further, there may be multiple semantics or meanings that explain or justify the existence of a set of links between the objects of two sources. A physical link between sources may correspond to multiple logical links. Thus, the problem of enumerating all paths that satisfy a query must be solved in an efficient manner.

There are five paths (without loops) starting from OMIM and terminating in PubMed. These paths are shown in Fig. 2. The value (in bold) associated with each path, represents the cardinality of *distinct* PubMed objects that were retrieved along each path, starting from the same set of OMIM objects; details of the experiment protocol to obtain these values are in [4]. As can be seen, there is a significant variation of the number of PubMed objects along each path. Object cardinality is a possible metric of the *Benefit* associated with a path. In this paper, we use two benefit metrics, Path Cardinality (PC) and Target Object Cardinality (TOC); other metrics have been discussed in [8, 10, 12]. Each path is also associated with a *Cost*, i.e., the cost of a query evaluation plan for the path. A plan for some path would be similar to a join query plan, except that forward and backward links between sources may not be symmetric; this is discussed later.

In [5], we presented an algorithm *ESearch*, based on a Deterministic Finite Automaton (DFA), to exhaustively enumerate all paths for some regular expression query representing a navigation. The challenge addressed in this paper is to develop heuristics that improve on the exhaustive *ESearch* and identify good *utility functions* that can rank the sources, the links between sources, and the sub-paths that are already visited, in order to quickly produce paths that

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright is held by the author/owner

Seventh International Workshop on the Web and Databases (WebDB 2004), June 17-18, 2004, Paris, France.

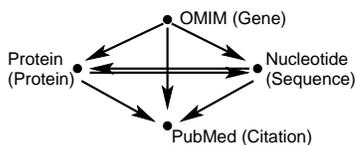


Figure 1: A source graph for OMIM and NCBI data sources (and corresponding scientific entities)

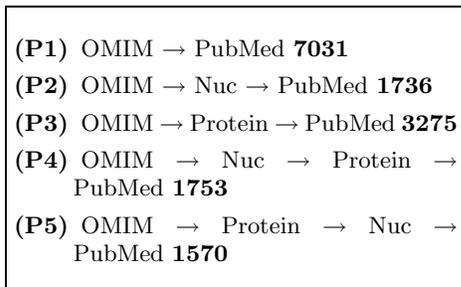


Figure 2: five paths from OMIM to PubMed

have the highest benefit and the least cost. Related problems have been presented in [1, 3, 8, 9, 10, 12].

In this paper, we present a heuristic that uses *local utility functions* to rank sources, using either the benefit attributed to the source, the cost of a plan using the source, or both benefit and cost. The heuristic will limit its search to some *Top XX%* of the ranked sources. To compare *ESearch* and the heuristic, we construct a Pareto surface of all dominant solutions produced by *ESearch*, with respect to benefit and cost. We choose the *Top 25%* of the *ESearch* solutions that are in the Pareto surface. We compare the paths produced by the heuristic to this *Top 25%* of *ESearch* solutions with respect to precision and recall. This motivates the need for further research on developing a more efficient algorithm and better utility functions to rank both the links between sources, and the sub-paths, so as to more efficiently produce better paths.

2. FINDING PATHS TO SATISFY REGULAR EXPRESSION QUERIES

2.1 Examples

Life science sources may be modeled at two levels: the physical and logical level. The physical level corresponds to the actual data sources and the links that exist between them. The physical level is modeled by a directed graph SG , such as illustrated in Figure 1, where nodes represent data sources and edges represent a physical implementation of a link between two data sources. A data object in one data source may have a link to one or more data objects in another data source, e.g., a gene in GeneCards links to a citation in PubMed. A path in SG is defined in a straightforward manner by traversing the links of SG . The logical level consists of classes (entity classes, concepts or ontology classes) that are implemented by one or more physical data sources or possibly parts of data sources. For example, the class *Citation* may be implemented by the data source PubMed. An example of a possible (commonly accepted)

mapping from logical classes to data sources is illustrated in Table 1.

CLASS	DATA SOURCE
Sequence (s)	NCBI Nucleotide database EMBL Nucleotide Sequence database DDBJ
Protein (p)	NCBI Protein database Swiss-Prot
Citation (c)	NCBI PubMed

Table 1: A Possible Mapping from Logical Classes to Physical Data Sources

Consider regular expressions expressed over class names in some set E . Given an input regular expression r , the objective is to interpret r on the graph SG . Class labels in E could include p (protein), s (sequence), g (gene), c (citation), etc., and ϵ (a wild card label). Each class label may have multiple interpretations in SG , e.g., sequence is interpreted by three sources in Table 1. We consider data sources including NCBI Nucleotide, EMBL Nucleotide Sequence, DDBJ, NCBI Protein, Swiss-Prot, HUGO, GeneCards and PubMed.

Regular expressions express common navigational queries. We note that a more expressive query language PQL has been presented in [7]. A scientist may be interested to *Retrieve all citations linked to proteins*; this is expressed as $p.c$. Class p can be interpreted by either NCBI Protein or Swiss-Prot. Class c 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 \rightarrow PubMed and Swiss-Prot \rightarrow PubMed, are solutions.

Consider a query *Retrieve citations linked to genes via any number of intermediate sources (paths of any length ≥ 2)*; it is $g.\epsilon^+.c$. This query will match any path that starts with a node that interprets g and terminates in a node that interprets c . The length of that path can vary from 2 to the length of the longest path in SG (if this path satisfies the expression). Thus, we see that enumerating all the paths in SG that match the query could be exponential in the size of SG . Such a query is important to scientists who wish to fully characterize scientific objects without specifying the sources and paths to visit.

Evaluating such a query is challenging since in order to obtain the most complete list of citations relevant to a gene, one would need to visit all relevant paths. On the other hand, in order to produce useful results quickly, or to be efficient in query evaluation, one may wish to visit the *best* paths, for example, *Top 25%* Pareto dominant paths.

2.2 ESearch: A DFA-based Exhaustive Algorithm

We consider a directed Source Graph $SG = (S, L)$. For this paper, we assume that each node of SG or a physical source implements exactly one entity class. Further, we assume that there is at most one edge between any two nodes of SG and that there are no cycles in SG . These assumptions will be relaxed in later research since a source often implements multiple entity classes and there are often multiple edges between sources. SG is defined as follows:

- S is a set of nodes where each node corresponds to a source.

- L is a set of edges $L \subset S \times S$ that represents links between sources.

The graph is mapped to the logical level as follows:

- ϕ is a mapping from a class name in E to a set of physical sources in S . ϕ is a one-to-many mapping as illustrated in Table 1. The wild card ϵ is mapped to S .
- γ is a (reverse) mapping from sources in S to class names in E . Note that each source of S is mapped by γ to a single class name in E , and to the wild card ϵ .

A path $p = (s_1, s_2, \dots, s_n)$ in SG is mapped to a regular expression $\gamma(p)$ defined by $\gamma(s_1).\gamma(s_2)\dots\gamma(s_n)$. The solution to a regular expression r over alphabet E is the set of simple paths p in SG that interpret r , that is $\{p \in SG \mid \phi(r) = p\}$. A simple path p may traverse the same source more than once, but not the same edge between 2 sources.

It was shown in [6] that for (any) graph and regular expression, determining if a particular edge occurred in a path that satisfied the regular expression and was in the answer, was NP complete. In [5], we present an algorithm *ESearch* that is based on a deterministic finite state automaton (DFA) that recognizes a regular expression. The algorithm performs an exhaustive breadth-first search of all paths in a graph. The DFA is represented by a set of transitions, where a transition is a triple $t=(i,f,e)$, where, i represents the initial state of t , f represents the final state of t and, e is the label of t and belongs to the set of entity classes E .

We briefly describe *ESearch*; details are in [5]. *ESearch* comprises two phases: (a) *build path* and (b) *print path*; we focus on *build path*. For each visited transition $t=(i,f,e)$, *build path* identifies all the sources $s_i \in \phi(e)$. If i is not a *start state* of the DFA, then, for each s_i , *build path* computes a set $s_i.previousSources$. To do so, it considers all the sources that were selected in transition t^p previous to t , and selects the subset of sources that are adjacent to s_i in SG ; these sources are included in $s_i.previousSources$. The *ESearch* algorithm runs in polynomial time in the size of the graph, if the graph is cycle-free and all paths are cycle-free. Each node (source) in the graph implements only one entity, so a node is visited at most once in each transition (each level of the breadth-first search). Similarly, each node is visited at most once in each iteration of *print path*. If d is the maximum number of sources that can precede a source in the annotated SG , i.e., the cardinality of *previousSources*, and b is the maximum length of (cycle free) paths satisfying the regular expression, then $O(b^d)$ is an upper bound for *ESearch*.

3. BENEFIT OF PATHS VERSUS THE COST OF EVALUATING A PLAN FOR THE PATH

Recall that in Fig. 2, there were five paths from OMIM to PubMed, and each path reached a different number of distinct PubMed objects, starting from an identical set of objects in OMIM. In the previous section, we discussed an algorithm to generate all the paths. We now consider how one may rank these paths. Thus, we discuss expressions to estimate the benefit of a path. We also consider the cost of a path, or the evaluation cost for a query to return answers after traversing this path.

3.1 Benefit of a Path

We focus on benefits that are based on statistics from SG and the data objects and links between objects in SG . One could also consider benefits that reflect the schema of SG , or specific objects, e.g., paths that include the source GenBank, or paths that include the specific gene TP53. We do not consider such benefits in this paper. We note that in most cases, the scientist may have some specific criteria to rank the benefit of paths. The following are some typical benefits and the objective may be to maximize (minimize) these benefits:

- Path length: The length of the path, or the number of distinct sources that are visited.
- Attribute cardinality: The (total) number of attributes of all (some) entity classes along the path.
- Result cardinality: The cardinality of the results or the number of retrieved entries. An object in a data source may be reached via multiple objects in some other data source, i.e., the target object may be a duplicate. Thus, we differentiate the *Path Cardinality (PC)*, i.e., the number of path instances in SG , and the *Target Object Cardinality (TOC)*, i.e., the number of distinct objects in the target (final) source of the path.

Consider a path p through sources S_1, S_2, \dots, S_n .

- Source cardinality: $c(S_i)$ is the number of data objects in source S_i .
- Link cardinality: $l(S_{i,i+1})$ is the number of links from all data objects of source S_i pointing to data objects of S_{i+1} .
- Link participation (start source): $l_{par}(S_{i,i+1})$ is the number of objects in S_i having at least one outgoing link to an object in S_{i+1} .
- Link image (target source): $l_{im}(S_{i,i+1})$ is the number of data objects in S_{i+1} that have at least one incoming link from objects in S_i .

Path Cardinality or PC Benefit:

The PC benefit for path p , $PC(p)$, where $n \geq 3$, is estimated as follows:

$$PC(p) = l(S_{1,2}) \times \prod_{i=2, \dots, n-1} [pcf(i)]. \quad (1)$$

If we assume uniform distribution of links and independence of an object having an incoming or outgoing link, then the path cardinality fraction is defined as: $pcf(i) = l(S_{i,i+1})/c(S_i)$. If we do not assume independence and assume that each incoming link is to an object with an outgoing link, then we have the following: $pcf(i) = l(S_{i,i+1})/l_{par}(S_i)$.

Target Object Cardinality or TOC Benefit:

The TOC benefit for path p is estimated as follows:

$$TOC(p) = c(S_n)tocf(n) \quad (2)$$

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 $tocf(2) = l_{im}(S_{1,2})/c(S_2)$.

We now derive an expression for $tocf(i+1)$ given some value for $tocf(i)$. An object x in S_{i+1} receives on average $\delta_{i+1}^{in} = l(S_{i,i+1})/l_{im}(S_{i,i+1})$ edges from objects in S_i . If at least one of these δ_{i+1}^{in} 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_{i+1}^{in} objects in S_i are not reached from some object in S_1 .

To compute $tocf(i+1)$ we compute the probability that an object is not reached for some $tocf(i)$, and then use that to compute the probability that it is reached. Thus, we have the following:

$$tocf(i+1) = (l_{im}(S_{i,i+1})/c(S_{i+1}))(1 - (1 - tocf(i))^{\delta_{i+1}^{in}}) \quad (3)$$

3.2 Cost of Evaluating a Plan for a Path

Consider a path p through sources S_1, S_2, \dots, S_n . The straightforward evaluation plan for this path is to start with source S_1 , traverse all links from S_1 to S_2 , and so on, until one reaches the target source S_n . However, one may also view this path as a join query between the corresponding sources, and thus, a plan to evaluate this path is the problem of finding a join order for this query.

Consider a link from S_j to S_{j+1} . Evaluating this link in the path can be implemented as a $hashJoin(S_{j,j+1})$ where objects from S_j and S_{j+1} are obtained, and a hash table is built. Note that the cost of the $hashJoin(S_{j+1,j})$ would be similar to the cost of $hashJoin(S_{j,j+1})$ since the hash table that is constructed is the same for both cases. An alternative is a navigational join ($navJoin(S_{j,j+1})$) where the objects of S_j are obtained and then, links to S_{j+1} are traversed to compute the join. Further, there may exist a reverse link from S_{j+1} to S_j . If these links instances are identical, i.e., all instances $l(S_{j,j+1})$ are identical to $l(S_{j+1,j})$, then the link is symmetric and the benefit of traversing it in either direction is the same. If the link is symmetric, then $navJoin(S_{j+1,j})$ is also an alternative. We note that 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.

We consider a greedy algorithm to determine the join order for the query (corresponding to the path p), and then we compute the cost of a plan for path p . First, we consider all alternative adjacent sources S_j and S_{j+1} in p and choose the pair with the lowest cost. In each subsequent iteration, we choose the pair with the lowest cost, assuming that all the joins of prior iterations have been computed, and that the objects have been obtained from those sources.

We determine the approximate cost of a plan obtained for path p , i.e., we approximate the local processing cost and the network costs to download objects. Using an approximate cost as we explore the search space is sufficient since in this process, we are only choosing among paths. We assume that there is a robust and accurate optimizer and a good cost model. Once a path is chosen, we can use the robust optimizer and the cost model to accurately estimate the costs and obtain the best plan for the path.

3.3 Cost and Benefit of the Paths Generated by ESearch

For this paper, we report on a single synthetic SG . The

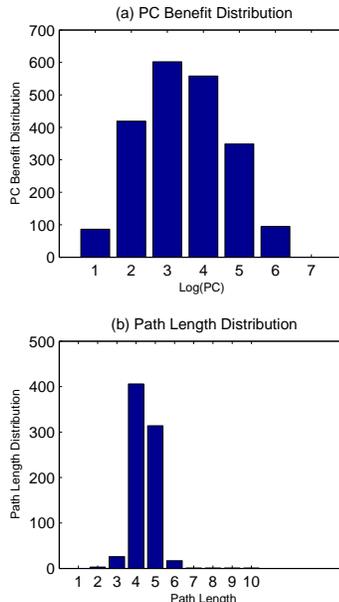


Figure 3: Path Length and Path Cardinality (PC)

metrics associated with the nodes and edges, i.e., the source and link cardinality, the link image and link participation, should reflect the diversity of life science sources and links. We generated an SG of 139 nodes and 1602 edges and varied the other metrics to reflect some typical distributions for these metrics. In future work, we will test the sensitivity of our heuristics to the diversity of these metrics.

Figures 3 and 4 illustrate the distribution of path length, the PC benefit, the TOC benefit, and the cost of plans for paths that are produced by $ESearch$. We report on the paths produced by $ESearch$ for 20 regular expressions. The unit for PC and TOC is the cardinality of objects. The Cost of a path is a relative cost of computational units and network transfer units; we note that in this paper we do not try to obtain exact evaluation costs for paths.

4. MULTI-OBJECTIVE OPTIMIZATION

Multi-objective optimization for database queries has been previously studied [1, 3, 8, 9, 10, 12]. The trade-off of execution cost versus delay in producing the results was studied in [10] in the context of the Mariposa wide area DBMS. More recently the trade-off of execution cost versus coverage was studied in [1, 8, 12] in the context of Internet sources with overlapping coverage. [3] 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 [8, 10, 12], in that they also developed heuristics to rank sources. However, they selected sources independently, and assumed all sources could answer the query (with different coverage or delay or cost). This is not true for our problem since a source may not always occur in a path that satisfies the regular expression, so source selection cannot be independent of the task of generating a path. While query optimization using limited capabilities [2, 11, 13] did consider that some sources may not result in *safe plans*, they did not consider dissimilar benefits of sources. While research in [1] does not directly address multi-criteria optimization, they

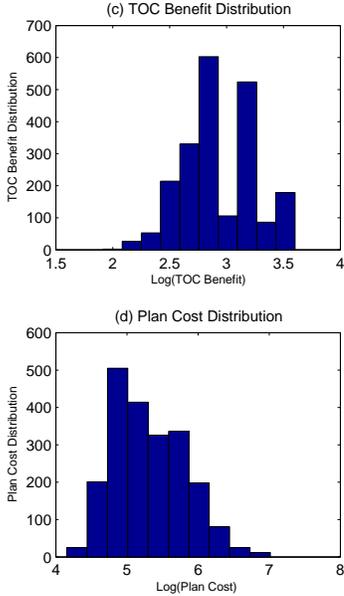


Figure 4: Target Object Cardinality (TOC) and Cost of Plan for Path

consider a variety of utility functions in choosing plans.

Our challenge is to develop heuristics that improve on the exhaustive *ESearch* and identify good *utility functions* that can rank the sources, the links between sources, and the sub-paths that are already visited, in order to quickly produce paths that have the highest benefit and the least cost. In the next section, we present a heuristic that uses a *local utility function* to rank sources, using the benefit attributed to the source, the cost of a plan using the source, or both benefit and cost. The heuristic will limit its search to some *Top XX%* of the ranked sources. We compare this heuristic with the *Top 25%* of the *ESearch* solutions that are in the Pareto surface to be described next.

4.1 Pareto Dominant Solutions and Top 25%

Given a set of paths, each characterized by a benefit and a cost, multi-objective optimization will try to find solutions that maximize benefit and minimize cost. An *dominant* path has the property that no other path dominates it for both benefit and cost. The set of all such *dominant* paths is a *Pareto surface or curve* for the paths. There has been much work on computing such curves. While there may be exponentially many paths on the Pareto curve, an *approximate Pareto curve* can often be constructed in polynomial time [9].

Recall that each of the paths generated by *ESearch* has either a PC or TOC benefit, as well as a Cost of the corresponding plan for evaluating the path. Our objective is to construct a set of *Relevant Paths*, say a *Top 25%* solution of a set of paths that is Pareto dominant. To define Pareto dominance of a set of paths, p_1, \dots, p_n , we consider the *Maximum Cost* of this set as the maximum Cost for each of the paths in the set. We also consider the *Maximum Benefit* as the *sum of either PC or TOC Benefit* for the set of paths. A Pareto dominant set of paths is one where there is no other set of paths which has *both* higher *Maximum Benefit* and lower *Maximum Cost*. In other words, we are interested in

minimizing the (maximum) cost and maximizing the (sum of the) benefit of the set of paths in the *Top 25% Relevant Paths*. We note that we could choose different definitions of the Pareto dominant set.

4.2 Local Utility Functions and a Heuristic to Choose OnlyTop XX% Sources

Recall that *ESearch* is a breadth-first search that exhaustively enumerates all paths for some query. Our objective is to develop a best-first search that can rank among sources and links between sources. A best-first search could also rank all currently explored sub-paths so as to quickly produce the best paths. Here, we present a heuristic to improve on *ESearch* by limiting *ESearch* to only explore *OnlyTop XX%* sources, in each step. This heuristic uses a *local utility function* as will be described.

The *OnlyTop XX%* heuristic modifies the *build path* phase of *ESearch*. As *build path* considers each transition t_k , all the sources satisfying transition t_k with label e in E are ranked based on a *local utility function*. From among all the sources $S_k \in \phi(e)$ that are also adjacent (have an edge in SG) to the sources chosen for the transition t_{k-1} previous to t_k , only those S_k that occur in the *OnlyTop XX%* of the ranking are chosen.

Four local utility functions are considered to rank the sources satisfying transition t_k . $lpc(S_k)$ and $ltoc(S_k)$ consider the (local) PC benefit and TOC benefit of source S_k . $lpcost(S_k)$ considers the local plan cost for source S_k to be joined to some source S_{k-1} that satisfies the transition t_{k-1} previous to t_k . Finally $lpc-lpcost$ considers both the benefit and the cost for source S_k . They are defined as follows:

- $lpc(S_k) = l(S_{k-1}, k)$.
- $ltoc(S_k) = ((1 - tocf(k-1))^{\delta_k^{in}})$.
- $lpcost(S_k) =$ the minimum of $hashJoin(S_{k-1}, k)$, $navJoin(S_{k-1}, k)$, and $navJoin(S_{k,k-1})$.

The *OnlyTopXX%* heuristic ranks the sources in descending order for the local utility function $lpc(S_k)$ or $ltoc(S_k)$ and in ascending order for $lpcost(S_k)$. The local utility function $lpc-lpcost$ only considers sources that are Pareto dominant based on PC benefit and Cost of plan; equal weight for PC benefit and Cost are used in the ranking.

5. CONCLUSIONS

We compare the precision (Figure 5) and recall (Figure 6) of the *OnlyTop XX%* heuristic compared to the *Top 25% Pareto* dominant solutions produced by *ESearch*. Recall that precision corresponds to those paths chosen by the heuristic which appear in the Pareto *Top 25%*. Figure 5 (a) reports on the precision as a fraction and Figure 5 (b) reports on the absolute number of paths that are generated by the heuristic. If we consider the precision, we see that using the PC benefit, Plan Cost, or the PC-Plan Cost trade-off utility function is usually better than using TOC benefit in choosing sources. As we increase from *OnlyTop 25%* to *OnlyTop 90%* sources, we see that the precision increases gradually for the case using the PC-Plan Cost trade-off, and it remains similar, or it may even decrease for the other utility functions. We also note that the absolute number of paths generated by the *OnlyTop 25%* heuristic, using any of the

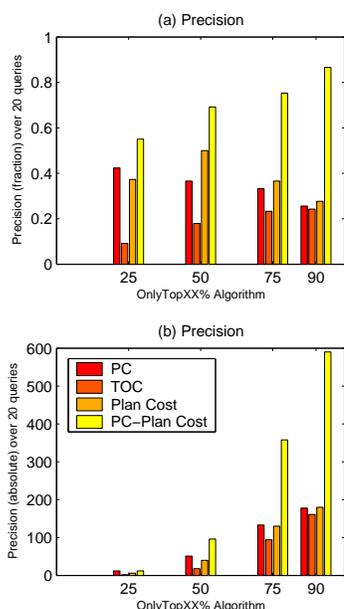


Figure 5: Precision of OnlyTop XX% Heuristic Search Compared to the Top 25% Pareto Dominant Solutions.

utility functions, is very small, since many of the sources that are selected do not result in safe paths that satisfy the regular expression.

To understand the situation better, we consider the recall in (Figure 6); the recall represents how many sources that are in the *Top 25% Pareto* are also generated by the heuristic. Here it is clear that even with 50% of the sources chosen by *OnlyTop 50%* heuristic (for any of the local utility functions), the recall is fairly low, e.g., around 20%. The recall only increases when more 75% of the sources are chosen, i.e., the heuristic will not significantly improve over the performance of the exhaustive *ESearch* algorithm which considers all the sources.

Our results show that PC benefit and Plan Cost and the PC-Plan Cost trade-off are reasonably good local utility functions to rank sources. However, in order to improve on *ESearch*, it is not sufficient to only consider local utility functions to choose sources. This motivates our future research to develop a best-first search that can rank among sources and links between sources. A best-first search should also rank all currently explored sub-paths so as to quickly produce the best paths.

6. ACKNOWLEDGMENTS

We thank J. Bleiholder, Z. Lacroix and F. Naumann for useful discussions.

7. REFERENCES

- [1] A. Doan and A. Halevy. Efficiently ordering query plans for data integration. In *Proceedings of the ICDE*, 2002.
- [2] D. Florescu, A. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proceedings of the ACM SIGMOD Conference*, 1999.

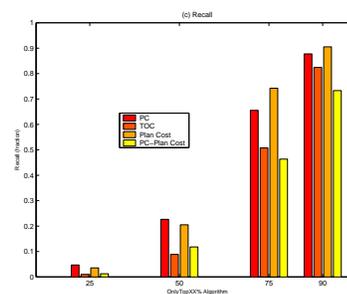


Figure 6: Recall of OnlyTop XX% Heuristic Search Compared to the top 25% Pareto Dominant Solutions.

- [3] 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.
- [4] Z. Lacroix, H. Murthy, F. Naumann, and L. Raschid. Characterizing properties of paths in biological data sources. *To appear in the Proceedings of the DILS Conference*, 2004.
- [5] Z. Lacroix, L. Raschid, and M. Vidal. Efficient techniques to explore paths in life science data sources. *To appear in the Proceedings of the DILS Conference*, 2004.
- [6] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 185–193, 1989.
- [7] P. Mork, R. Shaker, A. Halevy, and P. Tarczy-Hornoch. Pql: A declarative query language over dynamic biological data. *Proceedings of the AMIA*, 2002.
- [8] F. Naumann. *Quality-driven Query Answering for Integrated Information Systems*, volume 2261 of *Lecture Notes on Computer Science (LNCS)*. Springer Verlag, Heidelberg, 2002.
- [9] C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the ACM Symposium on Principles of Database Systems PODS01*, 2001.
- [10] 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.
- [11] M. Vidal. *Mediation Techniques for Multiple Autonomous Distributed Information Sources*. PhD thesis, Universidad Simón Bolívar, Caracas, Venezuela, 2000.
- [12] R. Yerneni, F. Naumann, and H. Garcia-Molina. Maximizing coverage of mediated web queries. *Stanford University Technical Report, Computer Science Department*, 2000.
- [13] V. Zadorozhny, L. Raschid, M. Vidal, T. Urhan, and L. Bright. Efficient Evaluation of Queries in a Mediator for WebSources. In *sigmod*, 2002.