Query Planning in the Presence of Overlapping Sources

Jens Bleiholder[1], Samir Khuller[2], Felix Naumann[1], Louiqa Raschid[2], and Yao Wu[2]

[1] Humboldt-Universität zu Berlin, Germany
{naumann, bleiho}@informatik.hu-berlin.de
[2] University of Maryland, College Park, Maryland, USA
{samir, yaowu}@cs.umd.edu, louiqa@umiacs.umd.edu

Abstract. Navigational queries on Web-accessible life science sources pose unique query optimization challenges. The objects in these sources are interconnected to objects in other sources, forming a large and complex graph, and there is an overlap of objects in the sources. Answering a query requires the traversal of multiple alternate paths through these sources. Each path can be associated with the benefit or the cardinality of the target object set (TOS) of objects reached in the result. There is also an evaluation cost of reaching the TOS.

We present dual problems in selecting the best set of paths. The first problem is to select a set of paths that satisfy a constraint on the evaluation cost while maximizing the benefit (number of distinct objects in the TOS). The dual problem is to select a set of paths that satisfies a threshold of the TOS benefit with minimal evaluation cost. The two problems can be mapped to the budgeted maximum coverage problem and the maximal set cover with a threshold. To solve these problems, we explore several solutions including greedy heuristics, a randomized search, and a traditional IP/LP formulation with bounds. We perform experiments on a real-world graph of life sciences objects from NCBI and report on the computational overhead of our solutions and their performance compared to the optimal solution.

1 Introduction

The last few years have seen an explosion in the number of public life science data sources, as well as the volume of data entries about scientific entities, such as genes, proteins, sequences, etc. Consequently, biologists spend a considerable amount of time navigating through the contents of these sources to obtain useful information. Life sciences sources, and the navigational queries that are of interest to scientists, pose some unique challenges. First, information about a certain scientific entity, e.g., a protein, may be available in a large number of autonomous sources, each possibly providing a different characterization of the entity. While the contents of these sources *overlap*, they are not replicas. Second, the *links* between scientific entities (links between data objects) in the different sources are unique in this domain in that they capture significant knowledge

Y. Ioannidis et al. (Eds.): EDBT 2006, LNCS 3896, pp. 811–828, 2006.

about the relationship and interactions between these entities. These links are uncovered in the process of navigation. Third, users are usually interested in navigational queries.

We consider a given set of sources and assume that the data objects in any of these sources have links to data objects in the other sources. We further assume that a (simple) navigational query identifies an origin class, e.g., protein, and possibly a (set of) origin sources that are of interest, e.g., UniProt. The query also identifies a target class of interest, e.g., publications, as well as an optional list of intermediate sources. Answering such queries involves exploring the data sources and classes, and the links between data sources. Our goal is to find paths at the logical level (among classes) and paths at the physical level (among sources implementing these classes). While we note that the query language can be extended to other query types, for our study we use a simple query.

Each path is associated with a *benefit*, namely the number of distinct objects reached in the target object set (TOS) in the target class. Each path is also associated with a *cost* of evaluating the query on the sources to compute the TOS. Given the overlap between sources and the highly interconnected nature of the object graph, each m-way combination of TOSs of paths is also associated with a *TOS overlap*. This overlap represents same objects reached in the TOS using different paths, and reduces the combined benefit of this path combination.

We present dual problems in this context of selecting the best set of paths. The first problem is to select a set of paths that satisfy a constraint on the evaluation cost while maximizing the benefit or the number of distinct objects in the TOS of these paths. This problem maps to the budgeted maximum coverage (BMC) problem [1]. We expect that in many cases, a user is more interested in reaching some desired minimal number of objects and may not set a constraint on the budget. To explore this situation, we consider the dual problem, which selects a set of paths that satisfies a threshold of the TOS benefit with minimal evaluation cost. The dual can be mapped to the maximal set cover with a threshold (MSCT).

The problems we address apply to many other scenarios. Consider a general problem - find a best set of paths to the data sources - and a simpler subproblem - find the best set of sources, ignoring that there might be multiple heterogeneous paths to reach these sources. This subproblem arises in many data integration situations, namely whenever (i) the integrated system has access to multiple sources that overlap in the data they store, (ii) it is not necessarily required to retrieve *all* answers to a query (some are enough), and (iii) some per-source cost is incurred to find and retrieve answers. Applications include metasearch engines and search engines for intranets, stock information systems (queries cost money), shopping agents, and digital libraries. For each of these systems it is worthwhile to access only some data sources and still find satisfying results.

Life science data sources are distributed web accessible sources. Consider the NCBI that is a portal providing access to all public NIH funded sources. For our research we created a warehouse of a subset of the links between 5 sources. We

note that creating a warehouse of links for all web access sources á la Google and providing the statistics needed to solve our problems is a difficult problem and is discussed in the experiment section.

Contributions. We identify a path overlap problem faced by life scientists in navigating paths among multiple interconnected and overlapping sources and we model it as a BMC problem and a dual MSCT problem. We propose an exact solution (IP), a randomized approximate solution with bounds (LP), unbounded greedy heuristics, and unbounded randomized solutions. Finally, we present empirical results of our strategies on a sampled real world data graph from the NCBI. The graph is stored in a database and we discuss the computational overhead supporting our solutions.

Outline. Sec. 2 first introduces our model of sources, objects, links, queries, and paths, and next formally states the two optimization problems. To solve the problems, Sec. 3 presents algorithms to compute exact solutions, an algorithm with known optimality bounds, and efficient but unbounded algorithms. Sec. 4 describes our experimental data of linked NCBI sources and their source-metrics, such as cardinality and overlap. In Sec. 5 we report on our experimental results showing good performance and solution quality for both problems. Finally, Sec. 6 reviews related work and Sec. 7 concludes.

2 Modeling Life Science Data Sources

We introduce a model for life science data sources and queries, and then define the problem of selecting sources and source paths to answer queries. Further and more detailed definitions are in [2].

2.1 Data Model and Queries

A scientific entity or class represents instances of a *logical class* of objects, e.g., Disease, Sequence, etc. A *logical link* is a directed relationship between two logical classes. The set of logical classes and logical links between them form the directed *logical graph LG*. A logical graph *LG* is an abstraction (or schema) of the source graph *SG* with data sources as nodes. A source *S* is a real-world accessible data source. Each logical class can be implemented by several sources. In turn, the object graph *OG* is an instance of *SG* containing representations of real-world objects and links between them. Finally, a result graph *RG* is a subset of *OG* and contains the data objects and links specific to a particular query. *LG*, *OG*, and *RG* are analogous to the schema, database instance, and result of a query.

Figure 1 shows an example of a world with four logical classes, Disease, Protein, Sequence, and Publication, and five sources. OMIM is the source that stores data on genetic knowledge on *Disease*, and Publications are stored in two sources PUBMED and BOOKS. Each source has some objects stored within, each having zero or more links to objects in other sources.

A source path p is a path from an *origin source* in the source graph SG to a *target source* of SG. Figure 2 lists the five source paths connecting origin source OMIM and target source PUBMED or BOOKS in our example SG.

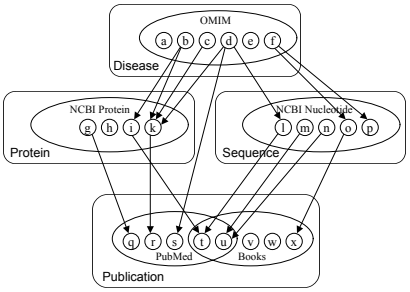


Fig. 1. A simple model with classes, sources, objects, and links

An *object link* is a directed edge between two data objects in two different sources. Given a source graph, the *object graph* OG is a directed graph in which the set O of all data objects stored by the sources are the nodes, and the set L of object links between these objects are the edges. The object graph represents our world model of all the objects and links that we consider. We note that object **t** in the OG of Fig. 1 occurs in the path overlap of two paths from OMIM to PUBMED, and in the path overlap of two paths from OMIM to BOOKS.

Consider the following query: *"Return all Publications of* PUBMED *that are linked to an* OMIM *entry about Diseases related to the keyword* `tumor`*."* To answer this query, a set of source paths in SG from OMIM to PUBMED are identified. A keyword search on `tumor` is used to retrieve relevant entries from OMIM. Then, for each source path, starting from the selected OMIM objects, all paths in OG that reach PUBMED entries are traversed. We define a result graph as representing the answers of a query against the OG. A result graph is a subset of the object graph. The *target object set (TOS)* is the set of objects in the RG reached in the target source of a particular source path. In the example the objects reached in PUBMED. Either the RG or the TOS for each of the source paths can be considered to be answers to the query. For the purposes of this paper, we consider simple queries that start with a set of objects in an origin source and traverse paths in OG to reach a set of objects (TOS) in each target source. More complex navigational queries are described in [3].

(P1) OMIM → PUBMED
(P2) OMIM → NCBI_Protein → PUBMED
(P3) OMIM → NCBI_Nucleotide → PUBMED
(P4) OMIM → NCBI_Protein → BOOKS
(P5) OMIM → NCBI_Nucleotide → BOOKS

Fig. 2. Five paths from OMIM to PUBMED or BOOKS through the source graph of Fig. 1

A source path p in SG can be characterized by a number of metrics, including the following:

- Length of the path
- Cardinality of attributes of all sources visited by p
- Cardinality of objects in the target source (TOS) - also called the benefit
- Cost of evaluating this source path on OG
- User's preference for objects in the TOS reached by traversing p

In related work we consider a multi-criteria optimization problem to generate the best K source paths or skyline paths [4].

2.2 Problem Statement

We formulate dual problems with respect to maximizing benefit given some starting object(s) and a cost constraint. The dual is to find a set of paths with benefit above a threshold that has the least cost.

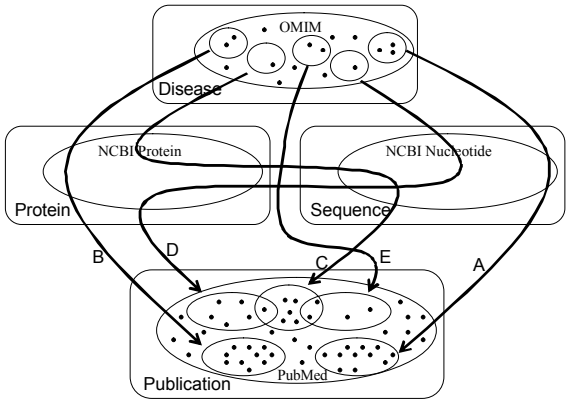
Consider the TOSs of two paths. We assume that the benefit of an object is 1. An important observation when counting target objects is that there is no additional benefit when the same object occurs in both TOS. We describe this as *TOS overlap*. TOS overlap can occur at different degrees, i.e., it can be disjoint, contained, equivalent, or have some concrete value. A discussion of overlap and methods to estimate overlap under certain assumptions is given in [2]. We note that while our problem definition assumes that there is no benefit of finding objects multiple times in overlapping paths, there are other contexts in which semantic knowledge is associated with overlap. For example, the fact that an object was reached by traversing two specific alternate paths may convey some knowledge about the characteristics of this object, or the sources involved in the paths.

We assume that there is a cost (or delay) associated with traversing the paths. This is realistic since accessing multiple sources may both delay the scientists as they wait for answers to be computed and delivered. It may also have a negative impact on all other users of these sources. Finally, the commercialization of certain data products means that actual payments may also be involved. A simple cost model would be to assign each path a unit cost (1). This turns the problem into choosing a combination of the best k paths among all possible paths. A more realistic way of assigning costs to the paths is to follow a cost model for query evaluation. In a later section, we discuss computing the metrics of paths in detail.

Assuming non-uniform costs, benefits, and TOS overlap, the problem is formally defined as follows:

Problem 1 (BMC). Consider a collection of paths $\mathcal{P} = \{p_1, p_2, \ldots p_m\}$, a world of objects $Z = \{z_1, z_2, \ldots z_n\}$ and a mapping to indicate if element z_j occurs in the TOS of path p_i. There is an associated cost for picking each path and an associated benefit for covering each element. Consider a collection of paths $\mathcal{P}' \subseteq \mathcal{P}$; the distinct objects in the corresponding union of the TOS for $\mathcal{P}'$ is labeled UnionTOS. The goal of our first problem is to find a set $\mathcal{P}'$ such that the total (adjusted) benefit of UnionTOS gained by picking $\mathcal{P}'$ is maximized, and the total cost of $\mathcal{P}'$ does not exceed a given budget B. The problem is known as the Budgeted Maximum Coverage (BMC) problem in the literature and is NP-hard [1]. Note that while the overall cost is the sum of the individual costs, the overall benefit is not the sum of individual benefits but must be adjusted (reduced) by any existing overlap.

Problem 2 (MSCT). The dual of this problem is the Maximal Set Coverage (with Threshold) or MSCT problem. The goal is to find a collection of paths



path	benefit	cost	ratio
A	11	20	0.55
B	10	13	0.77
C	8	10	0.8
D	6	10	0.6
E	4	6	0.67

Fig. 3. An example graph with five overlapping paths from Omim to PubMed and their benefit and cost

$\mathcal{P}' \subseteq \mathcal{P}$ such that the (adjusted) benefit of UnionTOS gained by picking $\mathcal{P}'$ is at least T while minimizing the total cost of $\mathcal{P}'$.

2.3 An Example

To illustrate how to choose combinations of paths we present an example. Figure 3 shows four data sources Omim, PubMed, NCBI_Protein, and NCBI_Sequence. There are five possible paths from Omim to PubMed (labeled A to E). Each path starts with a set of Omim objects and terminates in a set of PubMed objects. This results in the path benefit is also shown. The intermediate objects are not shown in Fig. 3 to not confuse the reader. The table of Fig. 3 shows cost and benefit/cost ratio.

Assume a cost limit of 20, which must be met and overlap as given in Fig. 3 (overlap(CD)=1 and overlap(CE)=2). We cannot follow all paths given the cost limit. We use this example in the following section to explain the different algorithms of finding the best subset of paths.

3 Algorithms

To solve BMC and MSCT, we implemented several algorithms to find a combination of best paths. After stating how to determine exact solutions for comparison purposes, we model the problem as an IP/LP, and then present some unbounded solutions including greedy algorithms and a random search algorithm. While the algorithms are applied to our small example here, we show how they perform on real-world data sets in Sec. 5.

3.1 Computing Exact Solutions

BMC: Determining an exact solution for BMC requires looking at all combinations of paths and their TOS, and choosing the one with the highest overlap-

adjusted benefit. The overlap-adjusted benefit simply eliminates duplicate objects in UnionTOS and then determines the cardinality. As the number of paths is exponential in the number of sources and the number of possible combinations is also exponential (2^n, n being the number of paths) we are dealing with a "doubly exponential" problem in the number of sources. However, due to the given cost limit, we may not need to consider all combinations. In our example only the combinations A, BE, CD, CE, and DE are below the cost limit of 20, the adjusted benefit results are 11, 14, 13, 10, and 10 respectively. We can see that the solution BE provides the best adjusted benefit of 14.

MSCT: Determining an exact solution for MSCT requires looking at all possible combinations of paths and their combined cost. The combination with the least cost is chosen, given that the overlap-adjusted benefit exceeds some threshold. In the example the maximum benefit one can gain is 36. If we want to find the cheapest solution with a guaranteed benefit of 29 (roughly 80%) only the combinations ABCDE (36), ABCD (34), ABCE (31), ABDE (31), and ABC (29) need to be considered (overlap adjusted benefit in parentheses). Among these ABC is cheapest with a cost of 43, still meeting the threshold.

Both problems can be modeled as Integer Programming to get an exact solution.

In summary, in both cases (BMC and MSCT) we are able to apply some pruning technique so that we do not need to consider all combinations and are able to speed up computation.

3.2 Formulation as an IP/LP

We solve BMC and MSCT using a standard LP relaxation and rounding approach. We show that the expected cost does not exceed the budget in BMC, and the expected benefit is within a factor of the optimal solution. We show that the expected benefit meets the threshold of the MSCT problem and the expected cost is at least within some factor of optimal. Interestingly, an almost identical randomized rounding approach is suitable for both problems as we show in [5].

BMC Problem: Let $\mathcal{S}$ be a family of sets (paths). Let $\mathcal{S} = \{S_1, S_2, \ldots S_m\}$ and let Z be the set of all objects, $Z = \{z_1, z_2, \ldots z_n\}$. Let B be the budget allowed to choose the subset of paths. We set integer variables $x_i = 1$ iff set S_i is picked and $y_j = 1$ iff z_j is covered. Let $c(S_i)$ be the cost of picking set S_i. w_j is the benefit of covering element z_j. In our problem, we consider a uniform benefit for all objects; that is, $w_j = 1$ for each object z_j. The IP formulation is as follows:

$$\begin{array}{lll} \text{maximize } \sum_{j=1}^{n} y_j \cdot w_j \text{ subject to } & \sum_{i=1}^{m} c(S_i) \cdot x_i & \leq B \\ & y_j & \leq \sum_{\{l | z_j \in S_l\}} x_l \text{ for all } j \\ & x_i \in \{0,1\} & \text{for all } i \\ & y_j \in \{0,1\} & \text{for all } j \end{array}$$

Although the IP gives an optimal solution to the problem, it is impractical to compute exact solutions; the IP problem is NP-complete. By relaxing the con-

straints that x_i and y_j must be integers, we have the following Linear Program (LP) formulation. Note that only the two last constraints of the IP formulation have been modified as follows: $x_i \leq 1$, $y_j \leq 1$.

We solve the LP (using the LP solver CPLEX) thus obtaining an optimal fractional solution, (x^*, y^*). We then choose a collection of sets $\mathcal{S}'$ such that $Pr[$ Set S_i is chosen in the set $\mathcal{S}'] = x_i^*$ by using a standard technique known as *randomized rounding* [6].

Algorithm. BMC_LP
- Solve the LP relaxation.
- Obtain fractional solution (x^*, y^*).
- Round x^* values to pick a subset of paths $\mathcal{S}'$.

This algorithm produces solutions whose expected costs do not exceed B and have an expected weight of the covered elements (TOS benefit) at least $(1 - \frac{1}{e})$ times the LP benefit [5]. Since the LP benefit is an upper bound on the optimal integral solution, this would be another way of deriving the bound developed earlier using a greedy algorithm combined with an enumeration approach [1].

MSCT Problem: The notation is the same as in the BMC problem, except that we want to choose a subset of paths that meet the threshold T while minimizing the cost. The IP formulation is as follows:

$$\begin{array}{lll} \text{minimize } \sum_{i=1}^{m} c(S_i) \cdot x_i \text{ subject to } \sum_{j=1}^{n} y_j \cdot w_j & \geq & T \\ y_j & \leq & \sum_{\{l \mid z_j \in S_l\}} x_l \text{ for all } j \\ x_i \in \{0,1\} & & \text{for all } i \\ y_j \in \{0,1\} & & \text{for all } j \end{array}$$

We relax the last two constraints in IP to obtain the LP formulation: $x_i \geq 0$, $y_j \geq 0$.

Let (x^*, y^*) be the fractional solution obtained by CPLEX. We choose a collection of sets $\mathcal{S}'$ such that $Pr[$ Set S_i is chosen in $\mathcal{S}'] = \min(1, \alpha x_i^*)$, where α is a boosting factor to ensure that we reach the threshold. This algorithm produces solutions with expected benefit at least $(1 - \frac{1}{e^\alpha}) \cdot T$ and expected cost at most $\alpha \cdot$ OPT [5].

3.3 Greedy Algorithms

We implemented several variants of a greedy heuristic and describe their evaluation in our experiments. Tab. 1 summarizes the results of all greedy algorithms for the example of Fig. 3. The choice of paths of each algorithm is indicated with a $*$ in Tab. 1.

Overlap-adjusted Greedy for BMC: Simple greedy variants (choosing paths in descending order of benefit or benefit/cost ratio) are not optimal, because the benefit considered does not take into account the overlap. In our example C and

E overlap by 2 so choosing C and E would give a benefit of only 10 instead of a benefit of 12. So the overlap-adjusted benefit should be taken into account in computing benefit to cost ratio. This strategy has been suggested in [1]. This requires some more computation as all benefits need to be adjusted in each step.

Algorithm. BMC_Greedy
- Rank paths by benefit/cost ratio, in descending order.
- Pick paths with largest benefit/cost ratio, adjust benefit/cost ratio of the remaining paths.
- Continue as long as the cost constraint (budget) is not exceeded.

In our example, path C is chosen first, as the benefit/cost ratio is highest. After choosing C the benefit/cost ratios are adjusted. As A and B cannot be chosen, because of exceeding cost limit, the algorithm chooses D next. This results in a solution of 93% (13/14) of the optimal solution.

Overlap-adjusted Greedy for MSCT: Similar to BMC_Greedy the greedy algorithm for MSCT also ranks the paths, but now by their cost/benefit ratio and the lowest ranked path at a time is chosen. The cost/benefit ratios of the other paths are adjusted. A solution is found as soon as the adjusted benefit of the combination meets the threshold. This threshold is equivalent to some fraction (e.g., 90%) of the maximum benefit possible, i.e., the overlap adjusted benefit if one chooses all possible paths. The algorithm finds a low cost solution guaranteeing a certain benefit (e.g., 90% of maximum benefit possible).

Algorithm. MSCT_Greedy
- Rank paths by cost/benefit ratio, in ascending order.
- Pick paths with smallest cost/benefit ratio, adjust cost/benefit ratio of the remaining paths.
- Continue as long as the benefit constraint is not met.

Assuming a benefit threshold of 29 (roughly 80%), path C is chosen first, as the cost/benefit ratio is lowest. Next, the ratios are adjusted and B and D are chosen, resulting in a partial solution with a cost of 33 and an adjusted benefit of 23. In a last step, the algorithm chooses A and reaches an adjusted benefit of 34 at a cost of 53. As the threshold of 29 is met, the algorithm stops with a solution of 123% (53/43) of the optimal.

Overlap-adjusted Greedy for MSCT with pruning: As one looks closer at the solution of MSCT_Greedy one finds that having chosen path D was a bad choice as even without it the benefit threshold also would have been met. Therefore we devised an improved version of the greedy algorithm, MSCT_Pruning, which, having chosen a combination of paths, reexamines all paths chosen and deletes one single redundant path, if one exists. If there is more than one redundant path, the path with the highest cost is deleted.

Table 1. Results of all greedy algorithms compared to the optimal solutions

path	benefit/cost ratio	BMC_optimal	BMC_Greedy	cost/benefit ratio	MSCT_optimal	MSCT_Greedy	MSCT_Pruning
A	0.55		(0.55)	1.81	*	*(1.81)	*
B	0.77	*	(0.62)	1.3	*	*(1.3)	*
C	0.8		* (n/a)	1.25	*	*	*
D	0.6		* (0.5)	1.67		* (1.43)	(*)
E	0.67	*	(0.33)	1.5		(3)	
achieved benefit		14	13		29	34	29
achieved cost		19	20		43	53	43

Algorithm. MSCT_Pruning
- Perform MSCT_Greedy.
- Pick each path of the combination, delete it; determine cost and benefit.
- Choose among these combinations the one with the smallest cost which also meets the threshold.

In our example, after having found the combination CBDA with a benefit of 34 and a cost of 53, the four combinations BDA (benefit 27, cost 43), CDA (benefit 24, cost 40), CBA (benefit 29, cost 43) and CBD (benefit 23, cost 33) are examined additionally. Combination CBA is chosen, as it meets the threshold at lowest cost.

3.4 Applying Randomized Optimization

We also applied a randomized technique to the BMC problem. Randomized approaches find solutions by searching guided by an utility function. The search through the search space involves random steps, in most cases resulting in faster convergence to a solution by leaving out unpromising parts of the search space.

Base algorithm. Goos describes different approaches to randomized optimization [7]. We use one specific specialization of the base algorithm, which is known as *Simulated Annealing*. Starting from an initial configuration K_0, new configurations are created involving the old configuration and some random decision. A new configuration is accepted if it is better than the old one, but it is also accepted with a certain probability if it is worse. The acceptance probability depends on the current *temperature*, lower temperature meaning lower acceptance probability. This enables the algorithm to escape local minima; escaping is likely in the beginning and becomes more and more unlikely. The temperature decreases over time, and the algorithm ends if the temperature drops below a predefined temperature, when it has "cooled down".

Modeling and implementation. We applied this random algorithm by modeling and changing configurations and minimizing an utility function. A configuration K to our problem consists of a set of paths, new configurations are created as follows:

1. Choose randomly among all available paths and combine them to a path set. This is done when initializing the start configuration K_0.
2. Add or delete a path. First, a path is chosen with a fixed probability out of all available paths and added to the set of paths if it is not already part of the set. Second, a path is randomly chosen out of all paths in the set and deleted from the set. This allows for inserting, deleting, and changing paths in the set. Creating a new configuration given an old one is done this way.

When changing a configuration (adding, deleting path), updating cost and benefit information could be done in $O(1)$, when using a bitset representation. When designing the utility function for BMC, overlap-adjusted benefit plays an important role, but also cost and other information could be used. Already a simple utility function consisting only of the overlap-adjusted benefit (OAB) and a penalty term for not complying with the cost limit yielded good results. The penalty term (MAX_COST) is set to a fixed number, exceeding the highest single path cost. The chosen utility function for BMC is shown in Equation 1.

$$UF_{BMC}(K) = \begin{cases} -OAB + MAX\_COST & \text{limit exceeded,} \\ -OAB & \text{otherwise} \end{cases} \quad (1)$$

3.5 Computational Complexity

LP: LP problem can be solved by Simplex algorithm in linear time "in practice". That is, the number of iterations is linear in the number of constraints, which is the total number of paths and objects. LP can also be solved in polynomial time by using Karmarkar's interior point method.
Greedy Algorithms: Given n paths and bitset representations of the paths and their objects, adjusting the benefit/cost ratio of a remaining path after having chosen a path could be done in constant time. Then all greedy variants have a computational complexity of $O(n^2)$.
Randomized Algorithm: The random approach has a computational complexity of $c*O(1)$. Here, c is a constant given by $c \leq a*b$ where a is the maximum number of configurations tested for acceptability per temperature and b is the number of distinct temperatures tried.

4 NCBI Data

4.1 NCBI Data Sources

NCBI/NIH is the gatekeeper for all biological data produced using federal funds in the US[1]. For the purpose of our experiments, we consider a source graph SG of five NCBI data sources (OMIM, PUBMED, NCBI_PROTEIN, NCBI_NUCLEOTIDE, and SNP), and the 10 links between these sources. We used the EFetch utility to sample all objects from these five sources that matched against

[1] www.ncbi.nlm.nih.gov

a set of several hundred keywords of interest. We then used the ELink utility to obtain all the links from these objects, along the 10 links, to the four other sources. We obtained an *OG* of approx. 28 million objects and 10 million links.

For simplicity, a query identified an origin source and a target source, and an optional keyword. A query is satisfied by up to sixteen source paths in the *SG* and is evaluated against the database of the sampled *OG*. For each of the source paths (and optional keyword), we determine the TOS; this is the set of objects reached in the target source. We also determine the cost of evaluating the TOS and the benefit (cardinality of the TOS).

4.2 Metrics for the NCBI Graph

We use a *bitset* data structure to store the TOS for each path and to compute the overlap of a set of TOS, and to store UnionTOS. UnionTOS is the union of a set of TOS(without duplicates). If an object (some position in the UnionTOS) is present in the path, the corresponding bit in the bitset vector for that path is set to 1. The bitset is used to efficiently compute the overlap adjusted benefit of a set of paths. We use DB2 union operator to help us to compute UnionTOS.

The IP/LP requires that the bitset for all paths must be computed a priori in order to set up the constraints of the IP/LP formulation. The greedy algorithm requires that some of the overlap adjusted benefits be pre-computed. While it does not require that the bitset be computed a priori, computing the bitset assists the algorithm. The random algorithm also computes the overlap adjusted benefit in an incremental manner and can benefit from the a priori computation of the bitset.

In general, the overhead of maintaining the desired metrics can be expensive. We briefly discuss some of the challenges. Consider computing the TOS or computing the benefit (cardinality of the TOS). Since we created a local database (warehouse) of all the objects that matched the keywords of interest, we could directly compute the TOS or its benefit. If the objects corresponding to the keyword were not sampled and stored in the relational database, we would have to *estimate* the TOS and its benefit. In prior work we have developed a model to make such estimations [2]. That model has the strong assumption of link independence, which may not hold for real sources.

Determining the cost associated with evaluating each search path on *OG* to compute the TOS is also straightforward in our case, because we consider only simple queries with a keyword of interest and we assume that the links of *OG* are stored in our relational database. In general, determining the cost of evaluating each source path involves estimating the cost of submitting EFetch queries to the NCBI servers to determine the objects that satisfied some complex search criterion, and possibly calls to ELink to find all objects that have links to an object of interest. It may also include some local join processing costs. The EFetch and ELink access cost depends on the workload on the NCBI servers and the network workload between the client and the NCBI servers. In our experiments, the cost associated with a path is the cost of computing the TOS of the path on the locally stored *OG*.

Consider for instance a query where the start source is PubMed and the target source is NCBI_Protein. As mentioned there are 16 source paths P0 through P15 between these two sources in this NCBI source graph; they visit the intermediate nodes NCBI_Nucleotide, SNP, or Omim. Note that the source paths do not have cycles and we do not visit a node or an edge more than once.

Table 2. Benefit of TOS and pair-wise overlap of 6 paths from PubMed to NCBI_Protein

	P0	P1	P2	P3	P4	P5
P0	30735	22729	2876	26	1560	166
P1		23916	1857	25	1573	108
P2			3261	24	1046	175
P3				40	37	21
P4					2848	80
P5						175

Table 2 reports on the TOS benefit for each path in the diagonal as well as the pair-wise TOS overlap between pairs of paths. For lack of space we report on only 6 of the 16 paths. As can be seen in Tab. 2, the TOS benefit for each individual path varies widely from 40 to 30735. We also note that the pair-wise TOS overlap between pairs of source paths has a wide variance of values and ranges from a low of 21 to a high of 22729.

We also illustrate the time to compute the UnionTOS and the time to compute the bitsets for a set of paths. We study 5 different queries, which induce different size of result graph. Thanks to DB2's union operator, we are able to compute UnionTOS for result graph of size hundreds of millions efficiently. Note that the time to compute UnionTOS is not proportional to the cardinality of the union, but depends on the inherent join complexity, that is cardinality of intermediate sources involved in the join. For example, consider the query from the origin source NCBI_Nucleotide to the target source Omim, even the cardinality of UnionTOS is relatively small, the join complexity is still comparable to rest queries.

Table 3. Running Time to Compute Metrics for a Large Object Graph and Result Graph

Query	size of UnionTOS	Time to Compute UnionTOS(msec)	Time to Compute bitset (msec)
NU to OM	8047	258095	609
NU to PU	122615	217412	7993
NU to PR	561358	164905	34689
PU to NU	1484403	282613	92329
NU to SN	1995918	502012	130765

The time to create bitsets for result graph is proportional to cardinality of UnionTOS as we expected. In the source graph we are interested in, where there are 5 sources and 10 links, the result graph can be computed efficiently. Computing these metrics for a large source graph could introduce scalability challenges and in the future work we will consider both specialized data structures and methods to estimate these metrics.

5 Experiments on NCBI Data

To demonstrate the effectiveness and efficiency of the different algorithms for the dual problems, we performed extensive experiments on different sampled real-world datasets. These are characterized by different start/end sources. For both BMC and MSCT we used a variety of budgets for cost and thresholds for benefit and compared the greedy, the random, and the LP solution to the exact solution. We first describe results for BMC, then for MSCT, and conclude with some remarks on their runtime.

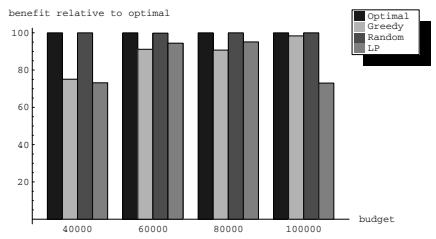
We used a total of 20 different start/end source combinations, but show results for only 2 characteristic ones; values are averaged over 10 samples. First we describe the experimental results, then we analyze them. The following figures all show relative solution quality compared to the optimal solution at varying budgets (BMC) and varying benefit thresholds (MSCT). Because BMC maximizes benefit, the algorithms do not reach 100% whereas MSCT minimizes cost and therefore the values are above 100%.

Results for BMC: Figure 4(a) shows results of experiments with all paths between sources SNP and PubMed with different cost limits. Algorithm Greedy is between 75% and 97% of the optimal solution, being worse at small budgets but with better relative results at higher budgets. The LP solution also lies between 75% and 95%, not showing improved performance with higher budgets, whereas the Random algorithm performs well for all budgets. The chosen path combinations (not shown here) consists of only a few paths. This leads to the difference in solution quality, as benefit may differ substantially if one single path is added/removed to/from the optimal solution.

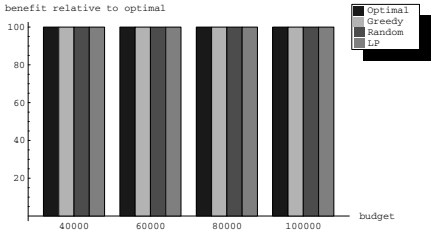
The results of experiments with all paths between source NCBI_Protein and PubMed are shown in Fig. 4(b). Here, all approaches perform exceptionally well, occasionally not finding the optimal solution but one at approximately 99.9% of it. Regarding the same path with smaller budgets in Fig. 4(c) shows something different: The same algorithms perform worse than with larger budgets (except Random). The reason for this behavior is that the budgets in the former case are so large that (almost) all paths are part of the solution. So the solution quality is influenced by the given budget.

Results for MSCT: Figure 4(d) shows results of experiments with all paths between source SNP and PubMed, with different benefit thresholds (0.7 meaning that the benefit of the solution is guaranteed to be at least 70% of the maximum possible benefit). Both algorithms perform well, Pruning being at least equal, but in most cases better than Greedy. At a threshold of 1.0 all paths must be chosen, except redundant paths. Greedy sometimes chooses these redundant paths and Pruning does not remove all, but only one. Therefore, both variants do not always find the optimal solution.

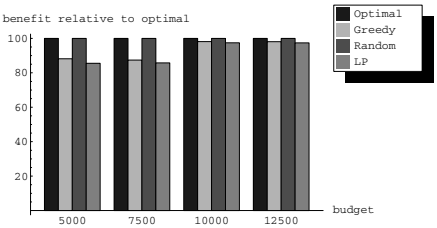
Discussion: Greedy seems to be the most unreliable algorithm among all. If the optimal solution is unambiguous (one path, all paths) it mostly finds it, but it has weaknesses in between. There is also a difference in solution quality, if two



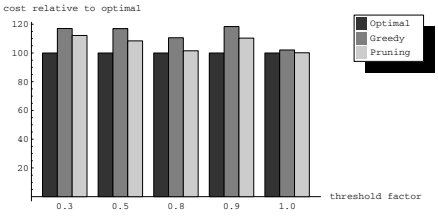
(a) BMC Algorithm Results for datasets from SNP to PubMed, large budgets.



(b) BMC Algorithm Results for datasets from NCBI_Protein to PubMed, large budgets.



(c) BMC Algorithm Results for datasets from NCBI_Protein to PubMed, small budgets.



(d) MSCT Algorithm Results for datasets from SNP to PubMed with different threshold.

Fig. 4. Selected experimental results

different datasets are compared. This seems to be accountable to the particular characteristics of the datasets. Comparing the solution quality of Greedy on the datasets SNP → PubMed (Fig. 4(a)) and NCBI_Protein → PubMed (Fig. 4(b)), Greedy performs better on the former. The difference in these two datasets lies in the cardinality and the spread of the overlap. In the sets where Greedy performs better, there is much more variation in the amount of overlap, ranging from just a few objects to paths that consist of 80% of all objects, including several fully contained other paths.

LP performance is comparable to that of Greedy. The differences in solution quality in different budgets may be explained by the randomized rounding approach: If LP happens to choose paths with a fractional value close to 1, and we set the budget as cut-off line, we have a good solution; but if LP chooses several paths with more or less equal probability around 0.5 (e.g., p1 with 0.51 and p2 with 0.52), then rounding favors p2 to p1. This may not be good in general.

In all tested samples Random performed very well. This is due to the inherent nature of randomized optimization: the randomness in considering different combinations of paths. Whereas Greedy deterministically determines a combination of paths, and may be misguided, Random chooses randomly, keeps good combinations and throws away poor ones. Performance also depends on a good utility function, which is very simple in our case but does a good job. It is

important to remark that we use one more or less straight forward parameter setting for all experiments. There is currently no need to adjust parameters to the characteristics of different data sets.

The fact that BMC and MSCT are coupled can also be seen in the results. In settings where Greedy performs better on BMC, it generally gives better results on MSCT, too. In that sense, characteristics of the data set in question influence the solution of both problems in a similar way.

One important decision in solving BMC is the choice of budget. From Figures 4(a) and 4(b) one can see that equal budgets result in different solution qualities, given different data sets. As different data sets result from different queries, we do not know in advance what quality the solution will preserve. In this sense, the MSCT problem is in fact the more interesting problem, as we can require a certain solution quality, as desired by scientists. By solving MSCT we gain information about the datasets, then we could make use of this knowledge and determine a budget for BMC if we have limited budget. This way the two problems interact and assist each other.

Runtime: Concerning average running time, the Greedy algorithms are the fastest among all algorithms, as expected. For all samples and all different budgets a solution is found within a few milliseconds on a state-of-the-art desktop computer. The runtime of Random is a few seconds on average whereas computing an optimal solution depends on the number of paths present in the data. In the real-world samples we tested, there are only 16 paths leading to an average runtime of a little less than a second, using the pruning technique mentioned in Sec. 3.1. However, determining exact solutions for worlds with more paths soon becomes infeasible. LP usually takes half a minute. However, since LP was run on a different platform, the runtimes are not directly comparable.

All algorithms would have longer running time if the computation of UnionTOS and the bitset were not completed a priori. The most significant performance impact is the LP formulation, which requires the complete computation of the bitset since it is needed to set up the constraints.

In summary, when an optimal solution is infeasible to compute, Greedy is the fastest while still returning good results. If one is able to invest some time, one should employ Random, as it gives nearly optimal objective values at reasonable runtimes. The advantage of LP is the guarantee (bounds) on the solution quality, albeit at a higher cost (higher runtime).

6 Related Work

Research in [8] and for Bibfinder [9] addresses the task of learning and maintaining statistics for distributed wide area applications. Bibfinder learns statistics for a variety of popular bibliographic data sources. Using a combination of machine learning and data mining techniques, it learns both coverage statistics of a source with respect to a query term (keyword) as well as the overlap among sources. We note that our task is more difficult, because we must consider the overlap of source paths, and the contents of all the sources occurring in the source path

may be associated (indexed) based on some query terms or keywords. Properties of links and paths have been studied in the context of XML document processing in the XSketch project [10], but the authors also do not consider overlap as a part of their framework.

While the optimization goal of conventional DBMS optimizers—find the complete and correct query result with minimal cost—is certainly different from the goal of this paper—find the most complete answer within a fixed budget—there are several aspects that carry over to our problem.

The first is *selectivity estimation* of query predicates, for instance as introduced in [11]. In our case, predicates are query keywords, which amounts to "="-predicates, and links between sources, which amounts to foreign key predicates. Based on certain assumptions, such as independence and uniform distribution of data values, selectivity estimators use table cardinalities and selectivity factors to estimate the result of joins and other operations. In our case, even the cardinality of base tables is difficult to assess. Currently we assume to simply have that information. Compounding the problem is the overlap of sources, which in effect reduces the cardinality-contribution of sources in some unknown way. The basics of selectivity estimation are used in our system, and future work will extend the metadata with histograms and will drop many of the underlying assumptions. Also, the advanced technique of learning statistics [12] is particularly useful in our scenario, as we are not merely dealing with approximate statistics but often with wholly unknown statistics of foreign sources.

A second similarity of conventional optimization lies in the *cost model*. Conventional optimizers usually model cost as processing time or throughput. In our scenario, the dominant cost is network traffic and the fetching of objects in other sources. Currently, we do not support different access paths, thus modeling the cost for us is straightforward. Again, future work can adopt a more sophisticated cost model making use of different ways to access data in sources, such as Web Services, JDBC, HTML forms, etc.

Finally, there is of course much to learn from distributed query processing [13]. As the query model presented in this paper is fairly simple and we focus on the logical aspects rather than the physical aspects of query execution, we do not discuss the relevance here. Our first priority is to provide functionality to biologists, our second is to provide it efficiently.

7 Conclusions and Future Work

Originally motivated by the problem of finding good paths and sets of paths through NCBI life sciences sources we have generalized the problem to data integration in the presence of overlapping sources, which applies to many different kinds of information systems. We presented a broad range of algorithms to solve this problem, from exact algorithms to bounded algorithms to greedy algorithms and simulated annealing. Each of these algorithms has different properties that we analyze and verify experimentally. To summarize, life sciences data sources are an excellent field to test new query models (paths through sources) and op-

timization problems (overlap-adjusted benefit), all the while solving problems that are relevant to biologists.

Possible future work is abundant. In a direct continuation of the work presented here we plan to expand on the types of queries to find out how our algorithms fare under different applications. Further strands of research are in the field of path query languages, efficient enumeration of all possible paths, and finally optimization techniques on the actual web-accessible NCBI sources rather than on large sampled sets stored in a local database.

Acknowledgment. This research was supported in part by the German Research Society (DFG grant no. NA 432) and NSF Grants IIS0205489, IIS0219909 and EIA0130422. We thank Maria Esther Vidal for helpful discussions.

References

1. Khuller, S., Moss, A., Naor, J.S.: The budgeted maximum coverage problem. Inf. Process. Lett. **70** (1999) 39–45
2. Lacroix, Z., Murthy, H., Naumann, F., Raschid, L.: Links and paths through life sciences data sources. In: Proceedings of the International Workshop on Data Integration for the Life Sciences (DILS), Springer (2004) 203–211
3. Mihaila, G., Naumann, F., Raschid, L., Vidal, M.E.: A data model and query language to explore enhanced links and paths in life science sources. In: Proceedings of the ACM SIGMOD Workshop on The Web and Databases (WebDB). (2005)
4. Raschid, L., Vidal, M.E., Cardenas, M., Marquez, N., Wu, Y.: Challenges of navigational queries: Finding best paths in graphs. Technical report, University of Maryland (2005)
5. Khuller, S., Raschid, L., Wu, Y.: LP randomized rounding for maximum coverage problem and minimum set cover with threshold problem. Technical report, University of Maryland (2005)
6. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge University Press (1995)
7. Goos, G.: Vorlesungen über Informatik - Paralleles Rechnen und nicht-analytische Lösungsverfahren. Volume 4. Springer Verlag, Berlin, Germany (1998)
8. Gruser, J.R., Raschid, L., Zadorozhny, V., Zhan, T.: Learning response time for websources using query feedback and application in query optimization. VLDB Journal **9** (2000) 18–37
9. Nie, Z., Kambhampati, S.: A frequency-based approach for mining coverage statistics in data integration. In: Proceedings of the International Conference on Data Engineering (ICDE). (2004) 387–398
10. Polyzotis, N., Garofalakis, M.: Structure and value synopses for XML data graphs. In: Proc. of the Int. Conf. on Very Large Databases (VLDB). (2002)
11. Selinger, P., Astrahan, M., Chamberlin, D., Lorie, R., Price, T.: Access path selection in a relational database management system. In: Proce. of the ACM Int. Conf. on Management of Data (SIGMOD), Boston, MA (1979) 23–34
12. Stillger, M., Lohman, G.M., Markl, V., Kandil, M.: LEO - DB2's LEarning Optimizer. In: Proc. of the Int. Conf. on Very Large Databases (VLDB), Rome, Italy (2001) 19–28
13. Kossmann, D.: The state of the art in distributed query processing. ACM Computing Surveys **32** (2000) 422–469