An Access Cost-Aware Approach for Object Retrieval over Multiple Sources

Benjamin Arai [#1], Gautam Das [*2], Dimitrios Gunopulos [$3], Vagelis Hristidis [+4], Nick Koudas [@5]

[#] *University of California, Riverside,* [*] *University of Texas, Arlington,* [$] *University of Athens, Greece*
[+] *Florida International University,* [@] *University of Toronto*
[1]barai@cs.ucr.edu, [2]gdas@cse.uta.edu, [3]dg@di.uoa.gr, [4]vagelis@cis.fiu.edu, [5]koudas@cs.toronto.edu

ABSTRACT

Source and object selection and retrieval from large multi-source data sets are fundamental operations in many applications. In this paper, we initiate research on efficient source (e.g., database) and object selection algorithms on large multi-source data sets. Specifically, in order to acquire a specified number of satisfying objects with minimum cost over multiple databases, the query engine needs to determine the access overhead for individual data sources, the overhead of retrieving objects from each source, and possibly other statistics such as estimating the frequency of finding a satisfying object in order to determine how many objects to retrieve from each data source. We adopt a probabilistic approach to source selection utilizing a cost structure and a dynamic programming model for computing the optimal number of objects to retrieve from each data source. Such a structure can be a valuable asset where there is a monetary or time related cost associated with accessing large distributed databases. We present a thorough experimental evaluation to validate our techniques using real-world data sets.

1. INTRODUCTION

Distributed object retrieval systems are becoming increasingly prevalent due to the growing size of data collections such as information logging, digitization of large periodicals (i.e., newspapers and books), and so on. Several information retrieval platforms have been developed including Google Bigtable [8] and OceanStore [4, 20, 19, 26]. These systems answer queries by retrieving objects from one or more sources (each source may have varying benefits). A key problem that such systems encounter is that many public interfaces have limited filters which cannot fulfill the user's request exactly. In this case, more data than is necessary to satisfy the query must be collected and then some filtering must be performed at the client side.

Large distributed collections present several distinct problems. Naïvely, a distributed collection of objects could be treated as a single monolithic database where each source would be independently accessed and the results from each source would be merged at the query node to create a single list of satisfying objects. There are several problems with this approach. First, the cost associated with accessing individual sources is not considered. Second, even if the costs were considered it is unclear how this could be done in a scalable and efficient (w.r.t., e.g., bandwidth and latency) fashion.

The problem of selecting how much to access each source is also important for single data source applications such as PubMed [25] and IMDB [18] (Internet Movie Database). As an example of a data source with an access cost model that involves both bandwidth and latency consider PubMed [25] (similar cost models can be defined for IMDB), a service of the U.S. National Library of Medicine that includes over 17 million citations (A citation is a reference to a published item, with sufficient detail to allow the reader to locate it.) from MEDLINE and other life science journals for biomedical articles dating back to the 1950s [25]. Suppose the user knows that there has been a study performed linking some genetic characteristic to a propensity for some type of cancer (e.g., breast cancer). The user's goal is to find the set of papers that mention the keyword "cancer" and also cite a paper that contains the phrase "genetic propensity" in its title. We can perform a keyword search for "cancer" but the query interface of PubMed does not allow keyword conditions on the citations (this is a limitation of the public interface). This query is not possible with the recent PubMed Advanced Search interface (http://www.ncbi.nlm.nih.gov/pubmed/advanced) either. Hence the client has to apply this extra filtering condition on the returned data. The question is: what is the best retrieval strategy in order to get five satisfying papers given that the query "cancer" returns more than two million papers? That is, how many objects should be retrieved at every iteration, given the access cost characteristics (access overhead, per object cost) of the source? In Section A we provide details on the access cost model of PubMed.

For the distributed (multi-source) scenario we must identify which source(s) to use for retrieval of objects and create an access strategy. This can be a difficult task when each source has a distinct access cost (e.g., latency, per-object monetary cost, etc.). This type of cost model is easy to see in modern information distribution sites. For instance, commercial legal research distributors (e.g., LexisNexis [21] & Westlaw [31]) offer a variety of data services based on several pay-per-search revenue models. The public interfaces of these databases allow a limited set of conditions to be specified (e.g., we may be able to specify a filing date range), but the client may be interested in further selection criteria (e.g., keywords such as "civil" and "riverside"). Such keyword searches are not allowed by these interfaces, hence the client has to query both databases and apply the extra selection criteria on the returned data. Using our approach, we can minimize the total access cost of such a query.

In our setting there are two tasks that we consider, specifically, *source selection* and *object selection*. Source selection decides which source(s) to retrieve objects from, whereas object selection decides how many objects to retrieve from each selected source.

VLDB '10, September 13-17, 2010, Singapore

These two tasks are interdependent and need to be addressed in a holistic way as we explain below.

When multiple sources are available for retrieval, a decision must be made on which source to access. Specifically, a decision needs to be made on which source will yield the highest benefit with minimal cost (overhead) for the user. Further, this cost may be in the form of money, bandwidth, and so on. Ideally, the system should address source selection in an automated fashion that considers source specific overhead and access costs. This system can then be used to respond to the users input while minimizing cost through accessing sources in an optimal fashion.

In order to select the optimal source for a given query, we need to determine the relevance of each source so that we can choose the one most relevant to a given query. Individual objects within a system could be classified as relevant (satisfying) or non relevant (unsatisfying). We adopt an *objective algorithmic* notion of relevance with a binary degree, following the principles defined in [6]: an object either meets the requirements of the query or it does not. Without loss of generality, a satisfying object can be described as an object that meets the requirements of a predefined query.

The fundamental component in multi-source object retrieval is the ability for a query to be submitted to the system and obtain k satisfying objects (viz., any-k) from a distributed database as though all the objects exist in a single, localized database. In order for such a process to be useful in a commercial setting, the solution must be both efficient and scalable. In this paper, we present an optimal approach utilizing dynamic programming to retrieve any k satisfying objects from multiple sources, minimizing cost (note: cost can be defined based on several criteria such as time and money), in a scalable manner to support both a large number of users and a large number of sources.

Our Contributions: The main contributions of this paper are summarized as follows:

- We introduce adaptive algorithms for source and object selection for the case when the expected frequency of satisfying objects is known. If this frequency is unknown, we propose techniques to adaptively learn it, which work well for sources where the retrieved objects represent a near-random sample of the database.
- We develop a probabilistic model for collections which contain various costs associated with accessing and retrieving objects from multiple databases.
- We provide an efficient solution for computing our algorithms utilizing pre-processing such that the per-query computational overhead is minimized.
- We demonstrate the efficiency and feasibility of our approach through a thorough experimental evaluation.

The rest of this paper is organized as follows. In Section 2 we present our framework. We introduce our approach in Sections 3 - 4. We provide extensive experimental evaluations of our technique in Section 5, and finally, in Section 6, we conclude this paper.

2. DATA MODEL & PROBLEM STATEMENT

In this section we discuss the principles behind our approach for cost-aware object retrieval. The relationship of the cost model to real data sources is discussed in Appendix A.

Our focus of this paper is to provide a probabilistic model for distributed collections which consider various economic costs associated with accessing and retrieving objects over multiple sources. Further, we introduce an a priori approach to computing our probabilistic model in such a way that the per-query computation cost is minimized. Providing an efficient and scalable approach is crucial in offering a competitive object retrieval system. Our goal is to compute these statistics as a pre-processing step, so that they are available for future queries.

A data source S_i stores a set $\{t_1^i, \ldots, t_{ni}^i\}$ of objects. The cost of retrieving l objects from S_i is $Cost_i(l)$. In this work we assume $Cost_i(l)$ is monotone (increasing) on l.

Assuming a public interface for each source, allowing a limited set of conditions, a client may be interested in further selection conditions not allowed by public interfaces. Therefore, the client must apply some extra selection condition on the returned data. In our setting, the interface to a source requests a number l of objects to retrieve. We represent this action as $GetNext_i(l)$. Note that when a source is accessed, the objects retrieval continues from where is stopped the previous time this source was accessed. This is a realistic assumption, given that a source access session is generally open for a long enough time for the source to maintain the current retrieval state.

We assume that we may have some statistics about the objects stored in a source. Techniques like [9, 10] can be used to infer such information. If the sampling is performed in a query-dependent way, then the cost of sampling should be added to the query cost. If the sampling is performed as a query-independent pre-processing step, then we do not add its cost to a query. We follow the latter in our experiments. We also present problem variants (SSNDD, MSNDD) where no source information is available and the algorithms adaptively learn the source probabilities.

The simplest way to model a source in terms of the frequency of satisfying objects is through the *query satisfying probability* $p_i(Q)$. Given a boolean selection query Q, the query satisfaction probability $p_i(Q)$ is the probability that an object retrieved from source S_i satisfies Q. Instead of the query satisfaction probability, we may have a query-specific histogram $H_i(Q)$ of the distribution of objects with respect to Q.

To setup the problem and develop our solutions we begin with the single source problem before continuing with the more challenging multiple source problem.

PROB 2.1 (SINGLE SOURCE, $p_i(Q)$ (SSQSP)). *Given any-k (retrieve any k results) query Q, data source S_i with access cost $Cost_i(l)$, and query satisfaction probability $p_i(Q)$, find best access strategy.*

By access strategy we mean a sequence of numbers l of objects to be retrieved from the source at each step. The best strategy is the one that minimizes the access cost. An interesting variant of the SSQSP problem is the case where there is no query satisfaction information. We call this variant Single Source, No Distribution Data (SSNDD) problem. As we explain later, for the SSNDD problem, our adaptive algorithms dynamically "learn" the source's distribution.

PROB 2.2 (SINGLE SOURCE (SSNDD)). *Given any-k query Q, data source S_i with access cost $Cost_i(l)$, and no query satisfaction information, find best access strategy.*

PROB 2.3 (MULTI-SOURCE, $p_i(Q)$ (MSQSP)). *Given any-k query Q, data sources $S_1, \ldots, S_q$ with access costs $Cost_i(l)$, and query satisfaction probabilities $p_i(Q)$, find best access strategy.*

As in the single source case, an interesting variant of the MSQSP problem is when there is no query satisfaction information. We call

this variant Multi-Source, No Distribution Data (MSNDD) problem.

PROB 2.4 (MULTI-SOURCE (MSNDD)). *Given any-k query Q, data sources $S_1, \ldots, S_q$ with access costs $Cost_i(l)$, and no query satisfaction information, find best access strategy.*

Example: Several online databases have emerged where there are costs associated with both accessing the database and retrieving information on a per-object basis as a means of generating revenue. There exists several revenue based *data discovery services* including LexisNexis [21] and Westlaw [31], which provide daily legal data such as civil litigation and bankruptcy filings.

Consider the following example, a trial lawyer is attempting to find 10 new cases but is paying for searches on each database separately. For each database there is a monetary cost for accessing a database and an additional cost for accessing objects (e.g., documents) from each. Suppose there are three databases, D_1, D_2, and D_3. We are also given some statistics for each database such as the cost of connecting and accessing individual objects for each database (a_1, a_2, and a_3) and (b_1, b_2, and b_3) respectively.

	D_1	D_2	D_3
Access Overhead (a)	300	1	100
Per-object Overhead (b)	1	1	2
Query Satisfaction Probability (p)	1%	0%	1%

Table 1: Sample source overheads, per-object costs and probabilities for sources D_1, D_2, and D_3.

Suppose a query engine receives a query to select 10 objects with the constraint *Q*="filing AND civil AND riverside" (i.e., Find 10 civil lawsuit filing which are in the county of riverside). The goal of the query is to locate 10 objects which satisfy the query Q accessing databases D_1, D_2, and D_3 such that the total cost is minimized (in this scenario sorting the objects in each database is not feasible since filings are continually being added).

Naïvely, we could retrieve objects from the database with the minimal access or per-object overhead but there are several problems with this approach. First, even if we choose the best database for retrieval, the chosen database may not be the optimal choice if the query is not satisfied in the first round. Second, other greedy approaches (i.e., doubling the retrieval size for each iteration until all satisfying objects are located) may be employed as a shotgun approach to retrieval but these types of techniques offer little benefit in the way of leveraging available statistic about the individual databases.

In contrast, we may be able to do much better if we consider the access overhead, per-object cost, and underlying object frequency (when available) when obtaining a satisfying object from each database. Using this information we can determine the optimal database to select for retrieval. The above table shows that database D_2 contains both a low access overhead and per-object cost but there are no satisfying objects in the source (Query Satisfaction Probability = 0%), so we can disregard D_2. This leaves us to consider D_1 and D_3.

Notice that D_1 contains a high access overhead but low per-object cost. However, D_3 contains a low access overhead but high per-object cost. In addition, D_1 has a p_1 of 1% of obtaining a satisfying object (i.e., sample about 1000 objects) and D_3 has a p_3 of 1% of obtaining a satisfying object (i.e., sample about 1000 objects). With this information we can compute the cost of retrieving 1000 objects from each database as $Cost_1(1000) = 300 + 1 \times 1000 = 1300$ and $Cost_3(1000) = 100 + 2 \times 1000 = 2100$. Clearly, D_1 is the better choice for the first round.

Symbol	Definition
Q	Query
k	Number of satisfying objects requested
a_i	Access overhead cost for source i.
b_i	Per-object overhead cost for source i.
q	Number of sources to retrieve data from.
S_i	A source used for retrieving data from.
p	Query satisfying probability for source S
n'	Number of satisfying objects retrieved thus far
l	The number of objects to retrieve in next step
$Cost(l)$	Access cost of source to retrieve l objects
$C(n', l)$	Cost of completing Q, given n' satisfying objects retrieved so far, and l objects will be retrieved in next step
$C(n')$	Cost of completing Q, given n' satisfying objects retrieved so far

Table 2: Symbol Description

Further, suppose in the first iteration we obtain only 9 of the 10 satisfying objects that we originally requested. We could retrieve additional objects from D_1 but it may no longer be the most cost effective source. Say we retrieve 100 object in the second iteration: $Cost_1(100) = 300 + 1 \times 100 = 400$ and $Cost_3(100) = 100 + 2 \times 100 = 300$ for databases D_1 and D_3 respectively. For the second iteration, D_1 is no longer the minimum cost source. Since we only need one satisfying object, a higher per-object cost can be tolerated as long as the access cost is sufficiently low making D_3 the ideal source for the second round. □

Other examples can be demonstrated which show additional ways which the optimal source may change during the retrieval process. Although the optimal source may change for each iteration, we show that our approach can adapt to varying constraints, consistently selecting the most desirable source and retrieval rates.

3. ALGORITHMS FOR THE SSQSP PROBLEM

In this section we present two algorithms to solve the SSQSP problem. The first one, Probabilistic Algorithm for SSQSP (P-SSQSP), uses a probabilistic argument to estimate the number of objects that should be retrieved in order to complete the query with a given probability (e.g., 95%).

The second algorithm, Dynamic Programming Method for SSQSP (DP-SSQSP), is optimal and is based on dynamic programming principles. P-SSQSP can be viewed as a greedy compromise to the optimal DP-SSQSP. We use the notation p instead of $p_i(Q)$ for simplicity. The two algorithms are experimentally compared in Section 5.

Appendix B shows how these algorithms are adapted for the MSQSP problem.

3.1 P-SSQSP: Probabilistic Algorithm for SSQSP Problem

In this algorithm, the number l of objects to be retrieved in the next step is computed in a way that the any-k query Q will be answered with probability $x\%$ by the end of this step. Let n' be the number of satisfying objects retrieved so far (i.e., $k - n'$ more satisfying objects need to be retrieved in the next steps).

The probability that exactly s satisfying objects are contained in the next l objects is computed using the binomial distribution as $\binom{l}{s} p^s (1-p)^{l-s}$.

Hence, the probability that Q is satisfied by retrieving l objects is given by the *cumulative binomial distribution*. This is the probability that at least $k - n'$ satisfying objects are retrieved, which is

$$P(Q\ completed) = \sum_{s=k-n'}^{l} \binom{l}{s} p^s (1-p)^{l-s} \quad (1)$$

We pick the minimum l such that $P(Q\ completed) \geq x\%$. That is,

$$l = argmin_l \left\{ \sum_{s=k}^{l} \binom{l}{s} p^s (1-p)^{l-s} \geq x\% \right\} \quad (2)$$

Note that l is recomputed at every step. Fortunately, since l is an integer and always positive we can efficiently compute the binomial coefficient for each value of l utilizing the Chernoff-Hoeffding inequality [17]. A key drawback of P-SSQSP algorithm is that the user must pick an x value, which plays a critical role in the performance of the algorithm. Intuitively, x should be higher for sources with high overhead cost, so that the probability of a subsequent retrieval step is small. In the following section (Section 3.2) we will explore the use of dynamic programming to take into account the cost incurred in the event that the next step does not complete the query.

$k - n'$	l
1	11
2	21
3	31

Table 3: Sample values for P-SSQSP algorithm for $p = 10\%$ and $x\% = 95\%$.

EXAMPLE 1. *Consider a source with $p = 0.1$, and let $x = 90\%$. Table 3 shows for given values of $k-n'$ (number of requested satisfying objects) the number l of objects to retrieve to complete the query.*

3.1.1 Space & Time Complexity

The space complexity of P-SSQSP is constant since we just need to evaluate the formula. Assuming the binomial coefficient can be computed in time linear in k (we can say this because we can compute the binomial coefficient incrementally from 1 to k):

$$\binom{n}{k} \times \frac{n-k}{k+1} = \binom{n}{k+1}$$

As shown above, each successive binomial coefficient is proportional to the current one. Using this property, the time complexity of P-SSQSP is $O(k \cdot l^2)$ per re-computation, which is expensive for large values of l. The time complexity can be reduced if a max value (l_{max}) for l is known (e.g., max access cost we are willing to pay, or size of source). Since we know from Equation 1 that the function is monotonically decreasing, we perform binary search on the values of l from $k+1$ to l_{max} until the appropriate l is found (setting probability just above $x\%$). This optimization provides an improved time complexity $O(l \cdot k \cdot log(l))$ per source access.

3.2 DP-SSQSP: Dynamic Programming Algorithm for SSQSP Problem

Overview: As mentioned above, the performance of P-SSQSP depends on the choice of x and there is no simple way to estimate a good value for it. We now present DP-SSQSP which is an optimal algorithm based on dynamic programming. The optimality is intuitively due to the fact that DP-SSQSP takes into account the cost incurred in the case that the next step does not complete the query (does not retrieve all any-k objects). Table 2 shows the symbols used.

Dynamic Programming Formulas: A key quantity in DP-SSQSP is the optimal expected cost $C(n', l)$ of completing the query Q, given that n' satisfying objects have been retrieved so far, and l objects will be retrieved in the next step. Obviously, $C(n', l) = 0$ if $n' \geq k$. To compute $C(n', l)$, we need to consider all possible scenarios, that is, possible numbers of returned satisfying objects.

Another key quantity is $C(n')$ (note the overloading of $C(.)$), which is the optimal expected cost to complete Q given that n' satisfying objects have been retrieved so far. It is

$$C(n') = min_{l \in I\!N} C(n', l) \quad (3)$$

We now go back to the calculation of $C(n', l)$. In order to account for all possible scenarios, the following formula is used.

$$C(n', l) = \sum_{s=0}^{k-n'} \big(P(s\ sat\ in\ l\ retr) \cdot C(n' + s) \big) + Cost(l) \quad (4)$$

where $P(s\ sat\ in\ l\ retr)$ is the probability that exactly s satisfying objects are contained in l retrieved objects.

$$P(s\ sat\ in\ l\ retr) = \binom{l}{s} p^s (1-p)^{l-s} \quad (5)$$

Recall that $Cost(l)$ is the access cost of the source to retrieve l objects (do not confuse $C(.)$ with $Cost(.)$).

Note in Equation 4 that the summation goes up to $k-n'$, because if more than $k-n'$ satisfying objects are returned the cost $C(n'+s)$ would be 0, since the query answer would have been answered.

A challenge in solving Equation 3 is that it is recursive. Since $C(n', l)$ is a function of $C(n')$ (the term in the sum for $s = 0$ in Equation 4). Hence, we need to find the $l \in I\!N$ value for which the following equation produces the minimum $C(n')$.

$$C(n') = \frac{\sum_{s=1}^{k-n'} \big(P(s\ sat\ in\ l\ retr) \cdot C(n' + s) \big) + Cost(l)}{1 - P(0\ sat\ in\ l\ retr)} \quad (6)$$

The terms $C(n' + s)$ for $s > 0$ are computed in the previous steps of DP-SSQSP algorithm and are hence treated as constants in Equation 6. The l value that minimizes $C(n')$ in Equation 6 can be computed either analytically, using derivatives (solve for $C(n')$, take derivative on l, set derivative equal to 0), or numerically. We use numeric methods in our experiments, which are more general. For the numeric methods to be efficient, we must assume that $Cost(l)$ is monotonically increasing with l, which is clearly a reasonable assumption.

Details: As in every dynamic programming algorithm, we need to fill up a table with interdependent values. As shown in Table 4, our dynamic programming table has the following columns: n', l, $C(n', l)$, $C(n')$.

We fill up the table (see Table 4) from top-down. The value in the bottom right cell is the expected optimal cost of the query execution. During query execution we read the table as follows: If we have retrieved n' satisfying objects so far, we use the lookup table to find the row with the minimum $C(n', l)$ value among the rows with the given n'.

THEOREM 1 (OPTIMALITY OF DP-SSQSP). *DP-SSQSP is an optimal algorithm for the SSQSP problem.*

n'	l	$C(n', l)$	$C(n')$
k	any	0	0
$k-1$	1	$(1-p) * C(k-1) + Cost(1)$	$min($
$k-1$	2	$(1-p)^2 * C(k-1) + Cost(2)$	$C(k-1,1)$,
$k-1$	3	...	$C(k-1,2)$,
$k-1$	...	...	...)
$k-2$	2	$(1-p)^2 * C(k-2) + 2p(1-p) * C(k-1) + Cost(2)$	$min($
$k-2$	3	$(1-p)^3 * C(k-2) + 3p(1-p)^2 * C(k-1) + Cost(3)$	$C(k-2,2)$,
$k-2$	4	...	$C(k-2,3)$,
$k-2$	...	...	...)
...	...	...	...
$k-k$	...	...	...

Table 4: Dynamic programming table created using DP-SSQSP algorithm.

The proof of this theorem follows from the fact that the dynamic programming algorithm considers the whole search space, and selects the optimal solution.

Pre-computing Lookup Tables: At query time, the only information we need is the optimal number l of objects to be retrieved in the next step, given the number $k - n'$ of remaining satisfying objects that need to be retrieved to complete the query. Hence, from the large table described above we only need to store the optimal l value for each $k - n'$, which is a k by 2 table, where k is the maximum number k of results that we expect a query to request. This table is called *lookup table*. To handle sources with different p values, we can store a lookup table for each p value (e.g., $p = 0.01, 0.02, \ldots, 1$). Then, at runtime, we only need to perform a sequence of lookups, instead of regenerating the whole dynamic programming table.

$k - n'$	l
1	42
2	91
3	144

Table 5: Lookup table for $p = 0.01$, $k = 3$, $Cost(l) = 10 + 1 \cdot l$.

EXAMPLE 2. *Table 5 shows the lookup table for a source with $p = 0.01$ and $Cost(l) = 10 + 1 \cdot l$. The table shows the optimal number l of objects to retrieve in the next iteration for various numbers of remaining satisfying objects to be retrieved to complete the query.*

Space & Time Complexity: We now present the space and time complexity of building the dynamic programming table shown in Table 4. Recall that the table is populated top-down. For each n' value, once we compute $Cost(n')$ we only need to keep the best l value (that minimizes $Cost(n', l)$) and $Cost(n', l)$. Hence, the space requirement for the minimization is $O(k)$.

The time complexity depends on the method used to find the l value that minimizes $C(n')$ in Equation 6. Naïvely, for each value of $Cost(n')$ we must compute the cost $Cost(n', l)$ for all possible values of l, we arrive at some l_{max} that is the number of points that were computed in order to find the minimum. This gives us a total time complexity of $O(k \cdot l_{max} \cdot (k - n'))$, however, in practice l_{max} for any given iteration, the optimal value for l tends to be very close to the value of l computed in the previous row of the dynamic programming table. This is easy to see, considering that for each row of the table the number of satisfying objects is increased by only one.

4. ALGORITHMS FOR THE SSNDD & MSNDD PROBLEMS

For the problems SSNDD (Single-Source, No Distribution Data) and MSNND (Multi-Source, No Distribution Data), where no information on the distribution of query satisfying objects is available, we modify our dynamic programming algorithms to "learn" the distribution from the objects retrieved during query execution. Note that this learning is only possible when the results of the query can be viewed as a random sample. This is generally not a realistic assumption, particularly when the objects' ranking for the query conditions ("cancer" in the example of Section 1) are correlated to the satisfaction condition (cite a paper that contains the phrase genetic propensity in its title). Nevertheless, we describe below an adaptation of the retrieval algorithms which works well in practice, as shown in Section 5, when the above assumption is reasonable.

In particular, we compute the current $p_i(Q)$ by performing a linear extrapolation of the objects retrieved to compute the satisfaction ratio. The challenge with this is picking an appropriate initial retrieval step l, given that no initial information is available for the probability $p_i(Q)$. In our experiments, where the source access cost function has the form $a + b \cdot l$, we use as a heuristic for the initial step l the ratio a/b. The rational is that we should make larger accesses for sources with high relative access overhead. If no satisfying objects are returned, we keep increasing the retrieval size (doubling the retrieval size for each round that zero satisfying objects have been retrieved). We stress here that there are alternative methods [1, 27, 24] for incrementally computing $p_i(Q)$ but these approaches are limited because they do not consider the case that the next step does not complete the query. We refer to the dynamic programming algorithms that learn the satisfaction probability as DP-SSNDD and DP-MSNDD.

5. EXPERIMENTAL EVALUATION

In this section we present an experimental evaluation of our approach. The implementation of our techniques is in GNU C++ and our evaluations were performed on a dual AMD Opteron 280 processor system with 4GB of memory running Linux. The index lists were stored in PostgreSQL[1] but for our experiments large blocks were fetched and cached for usage in the application. For comparison of our algorithms we offer several baseline approaches and two real-world data sets to compare our results. We chose the IMDB [18] (Internet Movie Database) data set because it is a well known and freely available and the Westlaw [31] set of legal documents because our approach is applicable to similar commercial data aggregation services. The multiple source problem is evaluated in Appendix D.

[1]http://www.postgresql.org

5.1 Data Sets

We have conducted a series of experiments using two real-world data sets: IMDB and Westlaw (new case summary reports for the state of California over a two-month period). We chose the IMDB and Westlaw databases because the entire data sets were available to us for analysis and experimentation. This allowed us to calculate exact values such as $p_i(Q)$ as described in Section 2.

For the IMDB data set we have gathered all of the titles[2] listed totaling 236,627 titles and for each object (i.e., TV show, movie, and so on) extracted a list of keywords describing each movie. This provides us with a database that can be used to perform a variety of searches such as locating specific genres and so on. For our experiments we have compiled 25 queries (which contained at least $k = 100$ satisfying objects for the purpose of experimental evaluation). Examples of these queries include: "based-on-novel", "kids-and-family", and "revenge". The frequency of the keywords used for our evaluation ranges from about 1% to 5%. For each of our experiments, we execute the algorithms for all keywords and take the average to display our results. For our experiments we retrieve objects from the IMDB data set in alphabetical order.

For the Westlaw data set we have gathered over 60,000 case filings for the state of California over a two-month period. These filings include several types of cases involving divorce, criminal litigation, and property disputes. Indexed keywords for this data set include a variety of fields including location of filings, case types, etc. For our experiments we compiled 25 keyword searches (which contained at least $k = 100$ satisfying objects for the purpose of experimental evaluation). Examples of these queries include: "finance", "copyright", and "trustee". The frequency of the keywords used for our evaluation ranges from about 0.1% to 1.0%. For each of our experiments, we execute the algorithms for all keywords and take the average to display our results. For our experiments we retrieve objects from the Westlaw data set in random order.

5.2 Experimental Setting

For all of our experiments we assume the cost is computed as $a + b \cdot l$. Unless otherwise specified we compute the exact binomial coefficient when generating the dynamic programming tables for each of our experiments. In order to evaluate our approach we offer several single-source baselines:

- **BAR (Base-AdaptStep-a/b):** retrieve a/b objects from the source for the first round, doubling[3] the sample size each round until k satisfying objects have been retrieved.
- **BAK (Base-AdaptStep-K):** retrieve k objects from the source for the first round, doubling the retrieval size each round until k satisfying objects have been retrieved.
- **BFR (Base-FixedStep-a/b):** retrieve a/b objects from the source as a fixed-step process until k satisfying objects have been retrieved.

Additional baseline approaches have been omitted from the experimental evaluation section because they performed at least an order of magnitude worse than the next competitive baseline approach.

As illustrated in Section 3, on-the-fly (at query time) computation of DP lookup tables may be slow. To address this issue we pre-compute DP lookup tables for several values of a, b, and p as a pre-processing step, as discussed in Appendix C.

[2]A title can be one of several types of titles such as TV show, movie, or documentary. Specifically, these titles are not limited to only movies.

[3]Doubling l is a standard technique used in approximation algorithms for fast convergence, and hence we also include them to have a complete study.

5.3 Object Selection Experiments

In the first set of experiments we explore the performance of our algorithms for determining the number of objects to select from a single source. In the following sections we will expand upon our object selection approach to include multiple sources (i.e., source selection).

Varying Access Overhead: In the first set of experiments we show how the DP-SSQSP, and DP-SSNDD algorithms perform in terms of cost defined in Section A, specifically, the total cost for completing a query including access overhead and per-object cost. In Figures 1, 2, and 3 we show that for the IMDB data set DP-SSQSP consistently outperforms both P-SSQSP and the baseline approaches. Further, when p is unknown at query time DP-SSNDD outperforms all of the baseline approaches (note that DP-SSNDD cannot be compared to P-SSQSP or DP-SSQSP directly since they require p to be known at query time). Similarly our algorithms perform well using the Westlaw database as illustrated in Figures 4, 5, and 6 achieving up to a 50% reduction in cost. For Figures 1 - 6 the cost function is defined to be $a + b \cdot l$ where $a \in 10, 50, 100$ and $b = 1$. Due to space constraints we have omitted experiments for varying values of b, but given the nature of the DP algorithms it should be clear that their performance is dependent upon the ratio of a to b.

It is important to note that for all of our experiments DP-SSQSP consistently outperformed the other algorithms achieving the lowest average cost. This suggests that there is a distinct benefit to considering a and b when determining the number of objects to retrieve in the next step.

The cost incurred by DP-SSNDD was on average higher than DP-SSQSP but for the majority of the experimental runs DP-SSNDD followed the performance of DP-SSQSP and always outperformed the baseline approaches. This suggests that even when p is unknown for the initial retrieval step there is sizable benefit to utilizing the dynamic programming tables. Also, DP-SSNDD outperforms P-SSQSP when the access cost is lower for the Westlaw data set.

Relative Error: Next we consider the relative error to evaluate the accuracy of our algorithms. We utilize the following formula to estimate the relative error for each of our experiments. First, we define *Optimal* as the optimal query cost. Assuming that the k-th satisfying result is at the s-th position ($s \geq k$), Optimal is cost of performing a single source access to retrieve s objects. $Cost$ is simply the cost incurred by an algorithm subject to the cost model defined in Section A. We can then evaluate the relative error for a query as:

$$RelativeError = \frac{(Cost - Optimal)}{Optimal}$$

As illustrated in Figures 7, 8, and 9, for various values of k our algorithms perform well using the IMDB data set. Specifically, DP-SSQSP outperforms the baseline approaches in all of our experiments achieving the lowest relative error (i.e., how much the query overshot the optimal retrieval size). Similarly as shown in Figures 10, 11, and 12, all our approaches consistently outperform the baseline approaches for each of the experiments using the Westlaw data set.

For both the IMDB and Westlaw databases the relative error for DP-SSQSP stays relatively flat between the values of k, this is due to the fact that taking into account the costs (i.e., a and b) helps reduce the relative error of the individual queries. Any of the baseline approaches may outperform the DP algorithms for a single query but on average the DP algorithms should outperform the baseline approaches. Notice that the DP algorithms consistently outperform P-SSQSP in the Westlaw data set, but not the IMDB data set. These

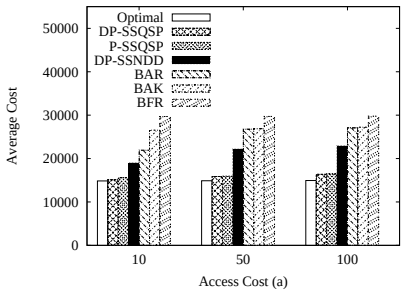

Figure 1: Comparison of average cost for various values of a. Each result is computed as the average cost over 25 queries where k=25 using the IMDB data set.

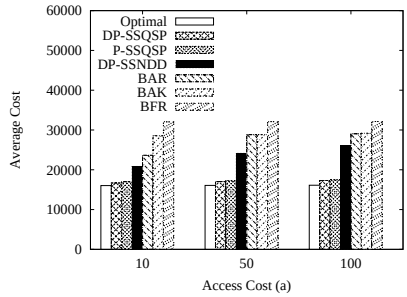

Figure 2: Comparison of average cost for various values of a. Each result is computed as the average cost over 25 queries where k=50 using the IMDB data set.

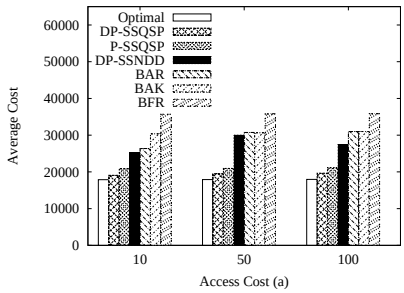

Figure 3: Comparison of average cost for various values of a. Each result is computed as the average cost over 25 queries where k=100 using the IMDB data set.

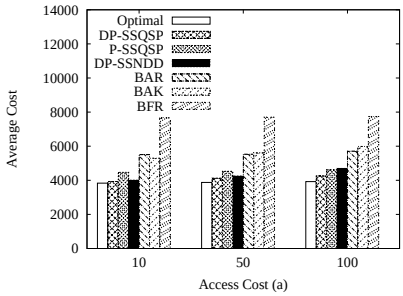

Figure 4: Comparison of average cost for various values of a. Each result is computed as the average cost over 25 queries where k=25 using the Westlaw data set.

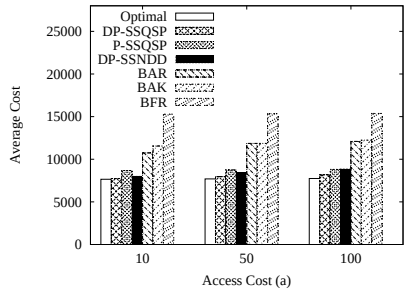

Figure 5: Comparison of average cost for various values of a. Each result is computed as the average cost over queries 25 where k=50 using the Westlaw data set.

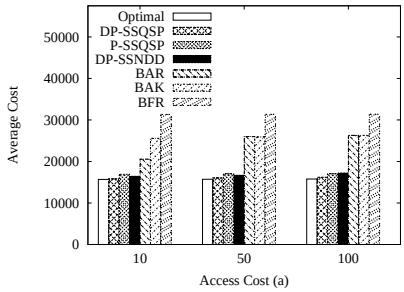

Figure 6: Comparison of average cost for various values of a. Each result is computed as the average cost over queries 25 where k=100 using the Westlaw data set.

results suggest that the algorithm is sensitive to the distribution of data. Specifically, queries performed on the randomly organized Westlaw data set tend to outperform queries on the alphabetically organized IMDB data set.

Varying Query Size: As illustrated in Figures 7, 8, 9, 10, 11, and 12 P-SSQSP, DP-SSQSP, and DP-SSNDD significantly outperform the baseline approaches using the IMDB and Westlaw data sets, consistently achieving a relative error half the size of the worst baseline approach.

In the case where p is unknown we can employ DP-SSNDD as described in Section 4. As illustrated in Figures 10, 11, and 12 DP-SSNDD achieves similar results to DP-SSQSP even though p is unknown for the initial step. As expected, DP-SSNDD performs well in comparison to the baselines, and outperforms P-SSQSP (recall p is available for P-SSQSP) as well. This suggests that ramp-up cost associated with DP-SSNDD does not greatly affect the overall performance of the algorithm.

6. CONCLUSION

In this paper, we have presented a cost aware approach to source and object selection, utilizing a cost structure and dynamic programming model for computing the optimal number of objects to retrieve from each data source. Such a structure can be a valuable asset where there is monetary or other costs associated with accessing large distributed databases.

We found that our approach excels in a variety of settings, such as when the number of objects requested is large and the frequency of retrieving a satisfying results is small. This can be explained by the fact that in such a scenario, blind doubling and other similar approaches do not fare well since the number of objects required may vary greatly. In contrast, we found that our approach does not greatly benefit object and source selection scenarios where the overhead was extremely small. This is understandable considering that as the overhead for both object and source selection approaches zero, it becomes increasingly desirable to employ greedy algorithms such as the doubling approach described earlier in the paper.

7. REFERENCES

[1] N. L. Arthur Dempster and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, page 39(1):138, 1977.

[2] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD Conference*, pages 28–39, 2003.

[3] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top k retrieval in peer-to-peer networks. In *ICDE*, pages 174–185, 2005.

[4] D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, and J. Kubiatowicz. Oceanstore: An extremely wide-area storage system. In *U.C. Berkeley Technical Report UCB//CSD-00-1102*, 1999.

[5] J. Bleiholder, Z. Lacroix, H. Murthy, F. Naumann, L. Raschid, and M.-E. Vidal. Biofast: Challenges in exploring linked life science sources. *SIGMOD Record*, 33(2):72–77, 2004.

[6] P. Borlund. The concept of relevance in IR. *Journal of the American Society for Information Science and Technology*, 54(10):913–925, 2003.

[7] J. P. Callan, Z. Lu, and W. B. Croft. Searching Distributed Collections with Inference Networks . In *SIGIR*, 1995.

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A

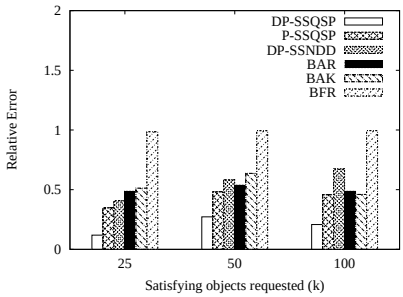

Figure 7: Comparison of the average relative error for various values of k. For each result the average error is reported over queries 25 where a=10 using the IMDB data set.

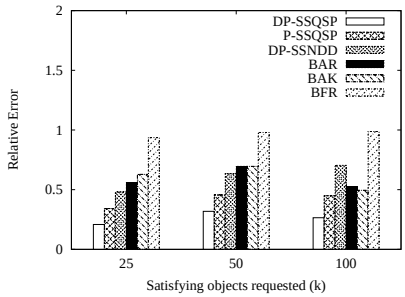

Figure 8: Comparison of the average relative error for various values of k. For each result the average error is reported over queries 25 where a=50 using the IMDB data set.

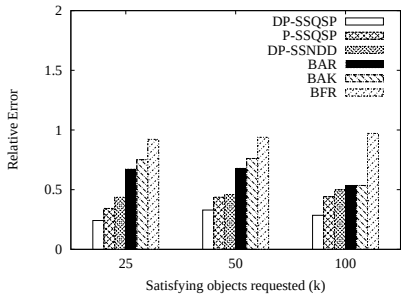

Figure 9: Comparison of the average relative error for various values of k. For each result the average error is reported over queries 25 where a=100 using the IMDB data set.

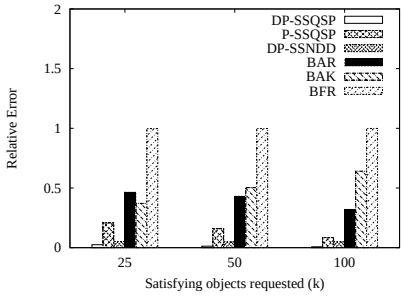

Figure 10: Comparison of the average relative error for various values of k. For each result the average error is reported over queries 25 where a=10 using the Westlaw data set.

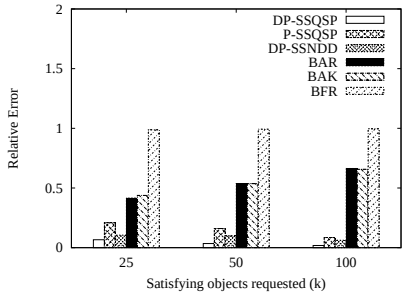

Figure 11: Comparison of the average relative error for various values of k. For each result the average error is reported over queries 25 where a=50 using the Westlaw data set.

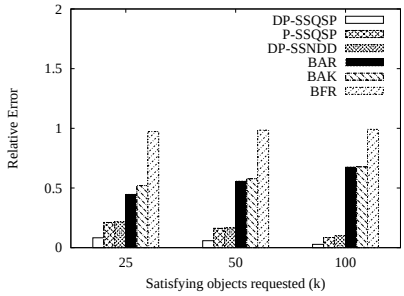

Figure 12: Comparison of the average relative error for various values of k. For each result the average error is reported over queries 25 where a=100 using the Westlaw data set.

distributed storage system for structured data. In *OSDI*, pages 205–218. USENIX Association, 2006.

[9] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. In *SIGMOD Conference*, pages 629–640, 2007.

[10] A. Dasgupta, N. Zhang, and G. Das. Leveraging count information in sampling hidden databases. In *ICDE*, pages 329–340, 2009.

[11] D. Dreilinger and A. E. Howe. Experiences with selecting search engines using metasearch. *ACM Transactions on Information Systems*, 15(3):195–222, 1997.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems*, 2001.

[13] N. Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–229, 1999.

[14] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The tsimmis approach to mediation: Data models and languages. *J. Intell. Inf. Syst.*, 8(2):117–132, 1997.

[15] L. Gravano and H. Garcia-Molina. Generalizing gloss to vector-space databases and broker hierarchies. In *VLDB*, pages 78–89, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[16] L. Gravano, H. García-Molina, and A. Tomasic. The effectiveness of gioss for the text database discovery problem. *SIGMOD Rec.*, 23(2):126–137, 1994.

[17] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[18] The Internet Movie Database (IMDB), http://www.imdb.org/.

[19] J. Kubiatowicz. Extracting guarantees from chaos. *Commun. ACM*, 46(2):33–38, 2003.

[20] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[21] LexisNexus. http://www.lexisnexis.com/.

[22] Z. Liu, C. Luo, J. Cho, and W. W. Chu. A probabilistic approach to metasearching with adaptive probing. In *ICDE*, pages 547–559, 2004.

[23] G. A. Mihaila, L. Raschid, and M.-E. Vidal. Source selection and ranking in the websemantics architecture: Using quality of data metadata. In *Advances in Computers 55: 89-119*, 2001.

[24] R. Neal and G. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, pages 355–368, 2005.

[25] PubMed. http://www.ncbi.nlm.nih.gov/pubmed/.

[26] S. C. Rhea, P. R. Eaton, D. Geels, H. Weatherspoon, B. Y. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *FAST*, 2003.

[27] J. M. Robert Hogg and A. Craig. Introduction to Mathematical Statistics. *Pearson Prentice Hall*, pages 359–364, 2005.

[28] M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum. Best-effort top-k query processing under budgetary constraints. In *ICDE*, 2009.

[29] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB*, pages 648–659, 2004.

[30] M.-E. Vidal, L. Raschid, and J. Mestre. Challenges in selecting paths for navigational queries: Trade-off of benefit of path versus cost of plan. In *WebDB*, pages 61–66, 2004.

[31] Westlaw. http://www.westlaw.com/.

[32] J. Xu and J. Callan. Effective retrieval with distributed collections. In *SIGIR*, pages 112–120, 1998.

APPENDIX

A. ACCESS COST MODEL

In this section we provide evidence supporting the existence of complex access cost models in real databases. In particular, we show that the access cost of real data sources generally consists of a relatively fixed overhead per access and a "per retrieved object" cost.

As stated in the introduction, our goal is to optimize access such that we minimize the total cost of retrieval for answering some query. In order to perform this optimization, we need to generate a cost model. Specifically, our model assumes that there is a distinct cost for accessing a source and a cost associated with retrieving individual objects. We break up the cost function into these distinct costs to handle a cost for connection to a database, and a cost for data transfer. This is motivated by real world databases which do not offer sufficient filters to answer a user's request—so the filtering must be performed on the client's side.

With every database connection there is an initiation "handshake" which establishes a connection between the client and the server. We represent the connection cost for each data source, S_i, as some constant cost, a_i.

We derive an expression for the transfer cost of a constant amount of data (i.e., the size of an object). Specifically, we can represent this cost as b. Further, the cost of retrieving l objects from a data source, S_i, is $b_i \cdot l$.

Separating the connection establishment cost from the data transfer cost allows us to apply the model to applications where individual sources contain unique connection establishment and data transfer costs. The cost function is defined to be,

$$Cost_i(l) = a_i + b_i \cdot l \tag{7}$$

where a_i is the access overhead cost and b_i is the per-object access cost: a_i is the cost of connecting to the data source (this may include authentication and other initiation procedures), b_i is the cost of retrieving individual objects from a data source. In approximating the retrieval cost with a_i and b_i, we also make the implicit assumption that the overhead of retrieving individual objects is similar for a local database.

An example of this model can be derived from the PubMed online database with the ESearch[4] online access system (searches and retrieves document ids, term translations), which is developed and maintained by National Center for Biotechnology Information (NCBI).

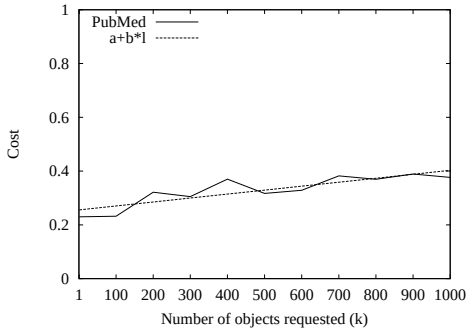

Figure 13: The average cost (seconds) to retrieve 1 to 1000 publication ids from the PubMed online database to satisfy the query "cancer". Each of the data points was averaged over 100 independent runs using a 1 Gb Internet connection.

[4]http://www.ncbi.nlm.nih.gov/entrez/query/static/esearch_help.html

As shown in Figure 13, the access cost for ESearch can be approximated by a linear function. We can compute the linear regression for the sample queries to estimate the constants in our cost model. There is a notable cost for connecting to the ESearch system (a = 0.2752 seconds). This can be accounted for by several possible underlying issues such as database connection initiation, network latency, etc. In addition, there is a small cost for retrieving individual objects (b = 0.0003 seconds). Note: similar results may be obtained using other access systems (e.g., Westlaw and IMDB), and values for a and b may vary depending upon several external influences such as user interaction and Internet connectivity.

This example demonstrates that our linear cost model can be applied to the commercially and publicly available systems. As well, it shows that the cost of overshooting the number of objects necessary to satisfy the query can become prohibitively expensive. Further, the ratio of a to b demonstrates that this is a non-trivial approach.

B. ALGORITHMS FOR THE MSQSP PROBLEM

In this section we consider the multiple source problem. To address this problem we have to develop solutions for the problem of source selection. We have sources $S_1, \ldots, S_q$ with query satisfaction probabilities $p_1(Q), \ldots, p_q(Q)$ respectively. We use the notation p_i instead of $p_i(Q)$ for simplicity. We assume in our analysis that the sources have disjoint objects. However, the algorithms are expected to perform well even for scenarios with reasonable overlap among sources. The only difference in the execution would be that the duplicate objects are discarded. If overlap information was available, it could be incorporated in our probabilistic model, and would generally lead to an increase of the retrieval step l.

B.1 P-MSQSP: Probabilistic Algorithm for MSQSP Problem

Here we develop a probabilistic algorithm for the MSQSP problem. The algorithm at each step computes the number l of objects to retrieve so that query Q is completed with probability $x\%$. However, the introduction of multiple sources introduces new constraints such as the minimum access cost to a source. We build on on the P-SSQSP algorithm, incorporating this idea as follows. At every step we compute separately the number l_i of objects to retrieve from each of the sources to complete the query with probability $x\%$. We pick the source S_i with minimum access cost $Cost_i(l_i)$ to do the next retrieval of l_i objects.

B.2 DP-MSQSP: Dynamic Programming Algorithm for MSQSP Problem

For DP-MSQSP a new constraint has been added; we must now also consider the extra source id dimension. For each step in the algorithm we must consider each source for selection. In particular, Equations 3 and 4 are modified as follows:

$$C(n') = min_{l \in I\!N, v \in 1,\ldots,q} C(n', l, v) \tag{8}$$

$$C(n') = \frac{\sum_{s=1}^{k-n'} \left(P_v(s \; sat \; in \; l \; retr) \cdot C(n'+s)\right) + Cost_v(l)}{1 - P_v(0 \; sat \; in \; l \; retr)} \tag{9}$$

where $C(n')$ is the expected cost of completing Q given that n' satisfying objects have been retrieved so far, and the next access will be at source S_v and will retrieve l objects.

The dynamic programming table as shown in Table 6 has an extra column v, which is the source id and goes from $1, \ldots, q$. The

n'	l	v	$C(n', l, v)$	$C(n')$
k	any	x	0	0
$k-1$	1	1	$(1-p_1) * C(k-1) + Cost_1(1)$	$min($ $C(k-1,1,1)$, $C(k-1,1,2)$, ..., $C(k-1,2,1)$, ..., $C(k-1,2,q)$, $C(k-1,3,1)$, ...)
$k-1$	1	2	$(1-p_2) * C(k-1) + Cost_2(1)$	
$k-1$	1	...	...	
$k-1$	1	q	$(1-p_q) * C(k-1) + Cost_q(1)$	
$k-1$	2	1...q	$(1-p_v)^2 * C(k-1) + Cost_v(2)$	
$k-1$	3	1...q	$(1-p_v)^3 * C(k-1) + Cost_v(3)$	
$k-1$	...		...	
$k-1$				
$k-1$				
$k-2$	2	1...q	$(1-p_v)^2 * C(k-2) + 2p_v(1-p_v) * C(k-1) + Cost_v(2)$	...
$k-2$	3	1...q	$(1-p_v)^3 * C(k-2) + 3p_v(1-p_v)^2 * C(k-1) + Cost_v(3)$	
$k-2$	4	1...q	...	
...	...			
0	1	1...q		Total cost of query
....	...			

Table 6: Dynamic programming table created using DP-MSQSP algorithm. The table shows how the first three values of n' are recursively computed.

lookup tables also have an extra column, the source id. That is, for each n' value, the lookup table stores the source id v, in addition to the number l of objects to retrieve from this source.

$k - n'$	l	v
1	208	3
2	235	2
3	305	2
4	361	1

Table 7: Sample lookup table for DP-MSQSP.

EXAMPLE 3. *Table 7 shows a sample lookup table for three sources with $p_1(Q) = 0.03, p_2(Q) = 0.02, p_3(Q) = 0.01$ and $Cost_1 = 100 + 0.01 \cdot l$, $Cost_2$=75+0.07·l, $Cost_3 = 50 + 0.1 \cdot l$ respectively. For instance, the second row shows that if two more satisfying objects need to be retrieved to complete Q, the next access should retrieve 235 objects from Source S_2.*

THEOREM 2 (OPTIMALITY OF DP-MSQSP). *DP-MSQSP is an optimal algorithm for the MSQSP problem.*

Space & Time Complexity: Similarly to the analysis of DP-SSQSP, space complexity is $O(k)$. We recall that $O(k \cdot l_{max} \cdot (k - n'))$ is the cost of solving the minimization problem of Equation 6, for multiple sources we simply extend our approach to consider all of the sources for each value of n', which gives us the following complexity $O(q \cdot k \cdot l_{max} \cdot (k - n'))$.

C. GENERATING DP LOOKUP TABLES

As illustrated in Section 3, on-the-fly (at query time) computation of DP lookup tables may not be desirable due to time constraints or other resource limitations. To address this issue we precompute DP lookup tables for several values of a, b, and p as a pre-processing step (the cost of computing a DP lookup table is described in Sections 3.2 and B.2), we can then use these precomputed tables for future queries with minimal per-query overhead (i.e., we only have to access the table containing the precomputed values). Given the numerous possible combinations for a, b, and p, computing a DP lookup table for all possible combinations can be prohibitively expensive.

For our experiments we compute DP lookup tables for various a, b, and p combinations. Since a and b are constant we compute tables only for the values pertinent to our experiments, but additional DP lookup tables can be created to accommodate sources with varying overhead. In contrast, p is not fixed (i.e., it may vary greatly for each query), so we must compute a sufficient number of DP lookup tables such that we adequately limit the error induced by rounding the exact p value to a pre-computed (discrete) value.

In our setting, for a given query the DP lookup table containing matching a, b, and p values is identified and used for retrieval. In the event that an exact match for a, b, and p has not been precomputed, we can estimate the number of objects to retrieve for the next step by using the entry with the closest a, b, and p values to the query. For our experiments, to ensure an adequate number of values for p are pre-computed we create several DP lookup tables: p between 0.1% to 25% in 0.1% increments (we also tested smaller increment sizes but we found that these additional DP lookup tables yielded little or no benefit for our experimental setting). This ensured that for any given query the pre-computed DP lookup table selected would not contain a deviation from the actual p of more than 0.1%. As stated earlier in the paper (Sections 3.2 and B.2) the storage requirement for DP lookup tables is reasonably small, the storage requirement for the pre-computed DP lookup tables is simply the number of DP lookup tables times $O(k)$. Appendix E shows the times needed to generate the DP lookup tables.

D. MULTIPLE SOURCE EXPERIMENTS

In this section we evaluate the multiple-source algorithms for the MSQSP Problem described in Appendix B. We use the following baselines:

- **BRFRM (Base-RoundRobin-Fixed-a/b-Multi):** randomly select an initial source, then retrieve objects in a round-robin fashion retrieving a_i/b_i objects from each source until k satisfying objects have been retrieved.
- **BRAKM (Base-RoundRobin-Adapt-K-Multi):** randomly select an initial source, retrieve k objects from the initial source for the first round, then in a round-robin fashion double the

retrieval size for each iteration until k satisfying objects have been retrieved.

To simulate an environment containing more than one source we have split the IMDB and Westlaw databases each into 10 partitions. Each partition represents a single source. The access overhead and per-object cost for each source was distributed as shown in Table 8.

Source	Access Overhead	Per-object Cost
1	1000	0.1
2	900	0.3
3	800	0.6
4	700	1.0
5	600	1.5
6	500	2.1
7	400	2.8
8	300	3.6
9	200	4.5
10	100	5.5

Table 8: Source access overhead and object selection cost used for each source for the multi-source experiments.

We have chosen the above access overhead and per-object cost for each source in order to maximize the number of the sources represented in the DP lookup table. It is easy to see that if the access overhead and per-object cost are not chosen carefully for each source, a few sources will dominate all of the entries in the DP lookup table.

For the multi-source experiments we show how DP-MSQSP performs in terms of cost, specifically, the total cost for completing a query including the access overhead and per-object costs across all sources accessed. Figures 14 and 15 demonstrate that our approach consistently outperforms the baseline approaches for various values of k using the IMDB and Westlaw data sets. In addition, DP-MSQSP consistently outperforms P-MSQSP. Similar to DP-SSQSP, the DP lookup tables allow for greater accuracy when determining the number of objects to retrieve over the less robust P-MSQSP, because costs are taken into account. For the multi-source experiments P-SSQSP and DP-SSQSP are very effective in keeping the cost low as k increases. Conversely, the baseline approaches experience a much more sporadic behavior. Results for DP-MSNDD have been omitted due to space constraints.

Number of Sources Utilized: In our experiments, when a source contains both a low access cost and a low per-object cost in comparison to other sources, it tends to dominate placement in the DP lookup table. Even when access costs and per-object costs across all sources are roughly equal, the dynamic programming algorithm tends to favor just a few sources for selection even when 10 or greater sources are available. This is understandable given the fact that there are two scenarios that need to be considered for source selection: 1) when the number of required satisfying objects is very large and 2) when few satisfying objects are needed. In the first case, a lower per-object cost is desired, and a higher access cost is acceptable. However, in the second case, a higher per-object cost can be tolerated as long as the access cost is sufficiently low.

For all of our experiments the number of sources utilized is never greater than two even when several sources are represented in the DP lookup table. For the majority of our experiments only one source is used even if multiple iterations are performed, this suggests that our dynamic programming algorithm does a reasonably good job at estimating the number of objects to retrieve.

E. EVALUATION OF DP LOOKUP TABLE CREATION

In our last set of experiments we explore the performance of our algorithm in terms of generating DP lookup tables. We note from Section 3.2 and Appendix B that we compute two types of DP lookup tables, namely, computing DP lookup tables for a single source and multiple sources (we assume 10 sources as in previous experiments). In Table 9 we show the performance of generating the DP lookup tables for k equals 100, 200, 300 using the technique for efficiently computing $P(s\ sat\ in\ l\ retr)$ as described in Section 3.1.

k	(1) Source DP Table	(10) Source DP Table
100	0.050	0.467
200	0.319	2.953
300	1.006	8.756

Table 9: Time (seconds) to generate dynamic programming tables for single and multiple source data sets.

A DP lookup table is created once for each query, and may be used for several queries thereafter. The results suggests that our approach can be effectively utilized for generating several dynamic programming tables as a pre-processing step with reasonable overhead.

F. RELATED WORK

Our problem is different from traditional data integration, where the query capabilities and schemas of various sources are combined. Instead, we assume that the problems of submitting a query to multiple sources [14] and locating the relevant sources [16, 15] are solved, and focus on the access strategy, i.e., the sequence and depth of accesses.

Several approaches have been presented for the *source selection* problem (i.e., which source(s) to use for sampling where features of individual sources such as probability of obtaining a satisfying object, cost of accessing a source, and cost of accessing individual objects may differ) [13, 16, 15, 32].

In [32] the authors employed the use of phrase information for *collection selection index* (complete list of objects and frequency for each) and *query expansion.* Both of these approaches offer increased discrimination capabilities across the different sources allowing for higher accuracy in choosing the most promising source. This approach is highly dependent upon a priori information and it is unclear how this information can merge efficiently between varying types of data sources without a significant overhead.

The best known algorithms which use frequency information for performing source selection are GIOSS [16, 15] and CORI [7]. CORI works by creating a *collection selection index* where each source is represented by its objects and individual object frequency. Each source is then ranked per query based on a object ranking algorithm. The GIOSS (Glossary-Of-Servers Server) system [16, 15] provides an approach for source selection by storing statistics of individual sources to estimate which source(s) may be the most useful for answering individual queries. In general, this approach uses term frequency information from individual sources to hint to the user which source may be the most beneficial for a given query. Mihaila et al. [23] present a framework to discover and combine Internet sources to answer complex queries, considering the Quality of Data and utility of the sources. However, these approaches do not address the problem of determining the optimal number of objects to retrieve from a selected source or subsequent accesses.

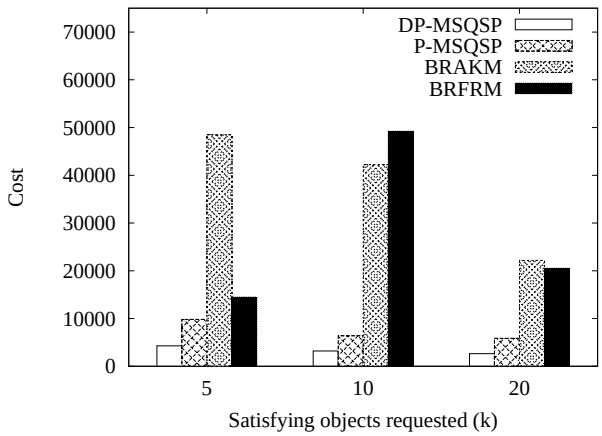

Figure 14: Comparison of the cost for the multi-source algorithms, averaging 25 queries for various values of k using the IMDB data set.

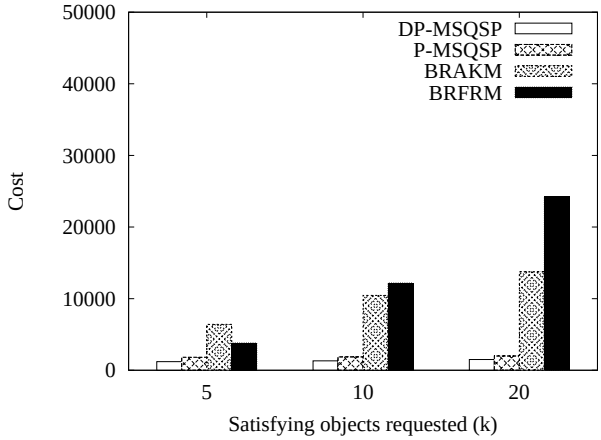

Figure 15: Comparison of the cost for the multi-source algorithms, averaging 25 queries for various values of k using the Westlaw data set.

Vidal et al. [30] propose a technique based on a local utility function to optimize the selection of paths in path queries on graph databases; this framework is introduced in BioFast [5]. The local utility function of a path is computed using its length, cardinality and other factors. Although our problem setting is very different, we can view the cost of accessing a source in our multi-source problem as a kind of local utility function.

Source and object selection is closely associated to *top-k query algorithms* [12] and *probabilistic top-k query algorithms* [29, 28] in traditional database research. Further, these approaches have been extended to distributed models answering queries on large multi-source databases [3, 2]. These approaches specifically focus on vertically partitioned (i.e., each source contains information about a specific attribute or group of attributes) data and it is not clear how these models can be applied to horizontally partitioned data in our distributed setting.

Recent work has explored the notion of probing databases to reduce the amount of work done by the user for source selection [22]. In essence, this approach probes (samples) individual sources developing a summary for each source. These summaries are then used to determine which source(s) are the most promising in satisfying the user's query. Further, approaches have been presented which utilize data collected from past queries to adaptively predict the utility of the individual sources [11].

Fuhr [13] presents an algorithm to compute the optimal number of documents to retrieve from each of a set of sources to answer an Information Retrieval query. Given the precision-recall graph of each source they pick a prefix of each source to retrieve to minimize the overall cost to get k relevant documents. However, their algorithm is optimal only if the precision-recall graphs are accurate, that is, it is not possible that a source will return a smaller than expected number of relevant documents. In contrast, our dynamic programming algorithm factors in this possibility since it just assumes a probability distribution of the relevant documents (satisfying objects in our setting).

G. ACKNOWLEDGMENTS

Gautam Das was supported by NSF grants 0916277, 0845644 and 0812601, a grant from Dept of Education, and unrestricted gifts from Microsoft Research and Nokia Research. Dimitrios Gunopulos was supported by NSF, and by the SemsorGrid4Env, Health-e-Child, and MODAP projects funded by the European Commission. Vagelis Hristidis was supported by NSF grant IIS-0811922 and DHS grant 2009-ST-062-000016.