Havasu: A Multi-Objective, Adaptive Query Processing Framework for Web Data Integration

Subbarao Kambhampati Ullas Nambiar Zaiqing Nie Sreelakshmi Vaddi*
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
{rao,mallu,nie,slakshmi}@asu.edu
ASU Technical Report TR-02-005

Abstract

Mediators for web-based data integration need the ability to handle multiple, often conflicting objectives, including cost, coverage and execution flexibility. This requires the development of query planning algorithms that are capable of multi-objective query optimization, as well as techniques for automatically gathering the requisite cost/coverage statistics from the autonomous data sources. We are designing a query processing framework called *Havasu* to handle these challenges. We will present the architecture of *Havasu* and describe the implementation and evaluation of its query planning and statistics gathering modules.

1 Introduction

The availability of structured information sources on the web has recently lead to significant interest in query processing frameworks that can integrate data sources available on the Internet. Some of the challenges involved in supporting such query processing–viz., the need to reformulate queries posed to the mediator into equivalent set of queries on the data sources, the ability to handle sources with limited access capabilities, and the need to handle uncertainty during execution time, have been addressed previously in [10, 6, 24, 22, 4, 14, 21, 9]. There are however two critical aspects of query processing in web-based data integration scenarios that have not yet been tackled adequately:

Multi-objective nature of query optimization: Unlike traditional data bases, where the objective of query optimization is solely to improve the query processing time, in data integration, optimization involves handling tradeoffs between multiple competing objectives. In addition to the familiar "execution cost", the dimensions of optimization also include: "coverage" of the query plan (i.e., the fraction of the answer tuples that the plan is estimated to provide), the "rate" at which the plan is expected to produce the answer tuples (c.f. [23]), and "execution flexibility" of the query plan (e.g. the ease with which the plan can be modified to handle execution time uncertainties such as blockages). Users may be interested

*Names listed in alphabetic order.

in plans that are optimal with respect to any of a variety of possible combinations of these different objectives. For example, some users may be interested in fast execution with reasonable coverage, while others may require high coverage even if with higher execution cost. Users may also be interested in plans that produce answer tuples at a steady clip (to support pipelined processing) rather than all at once at the end.

The various objectives are often *inter-dependent*–it is not possible to post-process a plan optimized for one objective (say "execution cost") to make it be optimal (or even "reasonable") with respect to the other objectives (e.g. "coverage" and "execution flexibility") [15]. The interactions between these competing objectives introduces several new challenges into query optimization [3, 19].

Challenges in gathering statistics: Effective query processing, especially one that is sensitive to multiple objectives, requires statistics about data sources. In addition to the familiar statistics such as selectivity estimates and relation cardinalities, in data integration scenarios, we also need statistics about source latency, transmission costs as well as the *coverage* and *overlap* information [8]. The last two are required for characterizing the distribution of the data in the underlying data sources. Unfortunately, gathering these statistics presents several challenges as data sources are autonomous. Furthermore, the need to manage time and space costs of statistics gathering becomes more critical, as the coverage and overlap statistics can be exponential in the number of data sources, and linear in the number of distinct queries (as in the worst case we will need overlap between every subset of sources with respect to a given query).

In this paper, we present *Havasu*, a novel framework for web-based data integration that we are currently developing, which focuses on multi-objective cost-based optimization. *Havasu* also explicitly tackles the challenges in gathering the requisite statistics to support the multi-objective query processing. *Havasu* adapts datamining and machine learning techniques to automatically gather relevant statistics about the data sources. As a first step towards this aim, we shall briefly describe how *Havasu* uses association rule mining techniques to gather statistics about the coverages and overlaps between the different sources. Our contribution here is a set of techniques that use hierarchical query classes and threshold-based variants of datamining techniques for keeping the time and space costs involved in statistics gathering tightly under control. *Havasu* uses the gathered statistics to support multi-objective query optimization. We will provide an overview of how *Havasu* currently generates plans that can jointly optimize cost and coverage of a query plan. Our contributions here include effective ways of estimating and combining the cost and coverage of partial plans, as well as developing query planning techniques that keep "planning time" (i.e. search for query plans) within reasonable limits, despite the increase complexity of optimization.

Architecture of *Havasu*: Figure 1 shows the architecture of *Havasu.* Like some previous data integration frameworks [13, 5, 12, 20, 21, 9, 1], *Havasu* assumes that the data sources can be modeled as constrained relational databases. *Havasu* uses the LAV (Local as View) approach to model the relation between the mediator and data source schemas. We assume that the source descriptions in terms of the mediator schema are provided as part of a "source registration" process. Once the registration is done, *StatMiner*, *Havasu*'s statistics gathering module, will learn the selectivity, response time, coverage and overlap statistics of the registered sources.

The user queries are posed on the mediator schema. The user is also given an opportunity to specify his/her tradeoffs between cost, coverage and time to first tuple in terms of a utility metric (see Section 3). The query and the utility metric are sent to *Multi-R*, the query optimizer module of *Havasu. Multi-R* will use the statistics provided by the *StatMiner* to support a multi-objective cost-based query optimization. The query plan produced by *Multi-R* is then "annotated" with the information about the execution costs

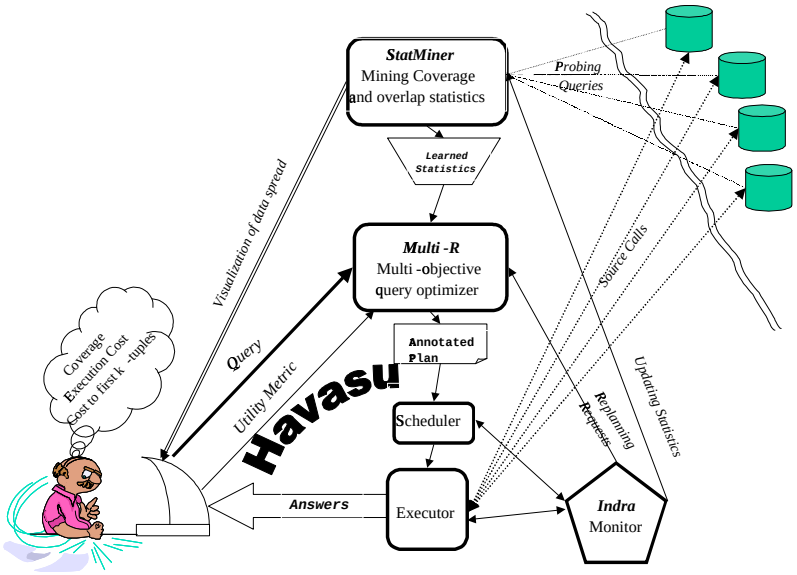


Figure 1: *Havasu* architecture

and coverages expected, and sent to the scheduler and executor. *Indra*, *Havasu*'s execution monitoring module, is responsible for detecting execution time deviations. It monitors the response times and coverages realized at various points in execution and compares them to the expected values as given by the plan annotations. When a discrepancy is detected, based on the type of discrepancy, *Indra* communicates with the executor and the query optimizer to either re-schedule or re-optimize the remaining part of the query plan. In some cases, *Indra* also communicates with *StatMiner* to update statistics to make them more accurate.

In this paper, we will describe the progress we made in designing *Multi-R* and *StatMiner* modules of *Havasu*. In the following two sections, we describe our current progress in implementing and evaluating these modules. We conclude, in Section 4 with a brief overview of some of the current directions of our research.

2 Mining statistics using *StatMiner*

As a first step towards fully automated statistics collection for data integration, we developed a set of techniques for learning coverage and overlap statistics for the data sources. *StatMiner* [16, 17] uses data mining techniques to automatically learn the coverage of autonomous sources, as well as their overlaps with respect to the global mediator schema. The key challenge in learning coverage statistics is keeping the number of needed statistics low enough to have the storage and learning costs manageable. The basic idea of our approach is to learn coverage statistics *not* with respect to individual queries but with respect to query classes. A query class is a set of (selection) queries sharing a particular set of features. We group queries into query classes using the *AV hierarchies* (or Attribute value hierarchies) of the attributes that are bound by the query. Specifically, we develop a hierarchical classification of queries starting with a hierarchical classification of the values of certain key attributes of the global relations. The current architecture of *StatMiner* is shown in Figure 2(a). The Mediator relations, description of the sources integrated by the mediator and the AV-hierarchies for each of the relations are to be provided by the designer of mediator and other domain experts. The mediator schema , AV hierarchies and source descriptions are used by *P*robing Query Generator to design and execute a set of probing queries. The results obtained

are further classified into the least general query classes that they belong to and stored in *c*lassInfo. The dataset *s*ourceInfo keeps track of what tuples were generated by each source. *StatMiner* then invokes the LCS algorithm [16] (see Learning Coverage and Overlap below) to identify the large mediator classes and the source coverages and source overlaps for them by using the datasets classInfo and sourceInfo.

Attribute Value Hierarchies: An *AV hierarchy* (or attribute value hierarchy) over an attribute A is a hierarchical classification of the values of the attribute A. The leaf nodes of the hierarchy correspond to specific concrete values of A, while the non-leaf nodes are abstract values that correspond to the union of values below them. Hierarchies do not have to exist for every attribute, but rather only for those attributes over which queries are classified. We call these attributes the **classificatory attributes**. Figure 2(b) shows sample AV hierarchies for two classificatory attributes Conference and Year of a relation **Paper**(Title, Author, Conference, Year). The selection of the classificatory attributes may either be done by the mediator designer or using automated techniques.

Query Classes: Since we focus on selection queries, a typical query will have values of some set of attributes bound. We group such queries into query classes using the AV hierarchies of the classificatory attributes that are bound by the query. To classify queries that do not bind any classificatory attribute, we would have to learn simple associations[1] between the values of the non-classificatory and classificatory attributes. A query class **feature** is defined as the assignment of a specific value to a classificatory attribute from its AV hierarchy. A feature is *a*bstract if the attribute is assigned an abstract (non-leaf) value from its AV hierarchy. Sets of features are used to define query classes. The space of query classes over which we learn the coverage and overlap statistics is just the cartesian product of the AV hierarchies of all the classificatory attributes. The AV hierarchies induce subsumption relations among the query classes. A class C_i is subsumed by class C_j if every feature in C_i is equal to, or a specialization of, the same dimension feature in C_j.

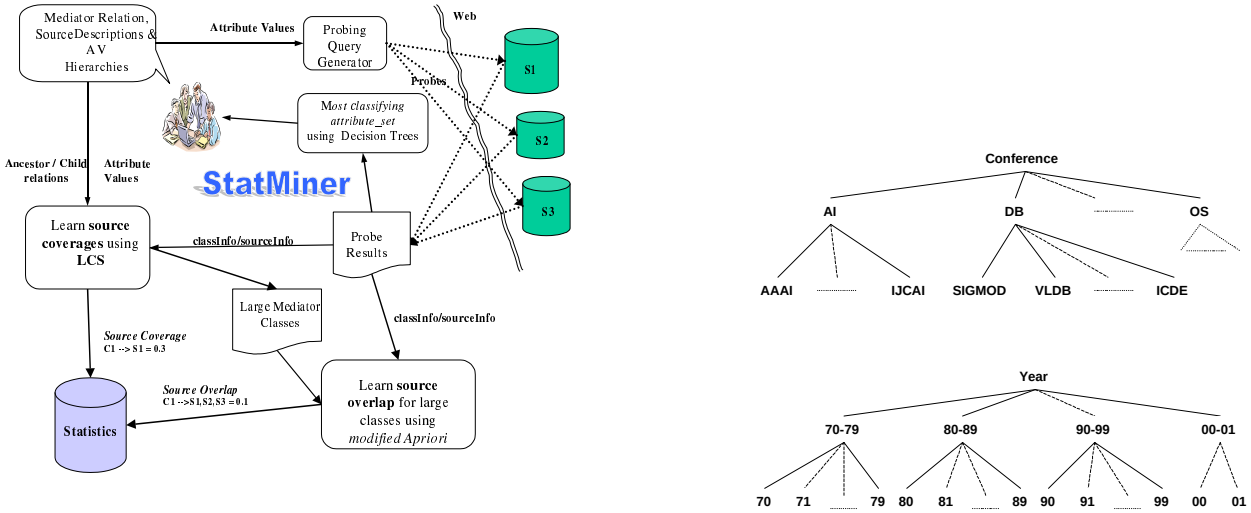
Source Coverage and Overlap: The *coverage* of a data source S with respect to a query class C, denoted by $P(S|C)$, is the probability that a random tuple belonging to the class C is present in source S. The *overlap* among a set $\widehat{S}$ of sources with respect to a class C, denoted by $P(\widehat{S}|C)$, is the probability that a random tuple belonging to the class C is present in each source $S \in \widehat{S}$.

Learning coverage and overlap statistics: The coverage and overlap can be conveniently computed using an association rule mining approach[2]. *StatMiner* learns coverage statistics using the LCS algorithm, and the overlap statistics using a variant of the Apriori algorithm [AS94]. LCS algorithm does two things: it identifies the query classes which have large enough support, and it computes the coverages of the individual sources with respect to these identified large classes. The resolution of the learned statistics is controlled in an adaptive manner with the help of two thresholds. A threshold τ_c is used to decide whether a query class has large enough support to be remembered. When a particular query class doesn't satisfy the minimum support threshold, *StatMiner* , in effect, stores statistics only with respect to some abstraction (generalization) of that class. Another threshold τ_o is used to decide whether or not the overlap statistics between a set of sources and a remembered query class should be stored. A detailed description of LCS

[1] A simple association would be $Author = J.Ullman \rightarrow Conference = Databases$ where $Author$ is non-classificatory while $Conference$ is a classificatory attribute

[2] Support and confidence are two measures of a rule's significance. The support of the rule $C \rightarrow \widehat{S}$ (denoted by $P(C \cap \widehat{S})$) refers to the percentage of the tuples in the global relation that are common to all the sources in set $\widehat{S}$ and belong to class C. The confidence of the rule (denoted by $P(\widehat{S}|C) = \frac{P(C \cap \widehat{S})}{P(C)}$) refers to the percentage of the tuples in the class C that are common to all the sources in $sourceSet$ $\widehat{S}$.

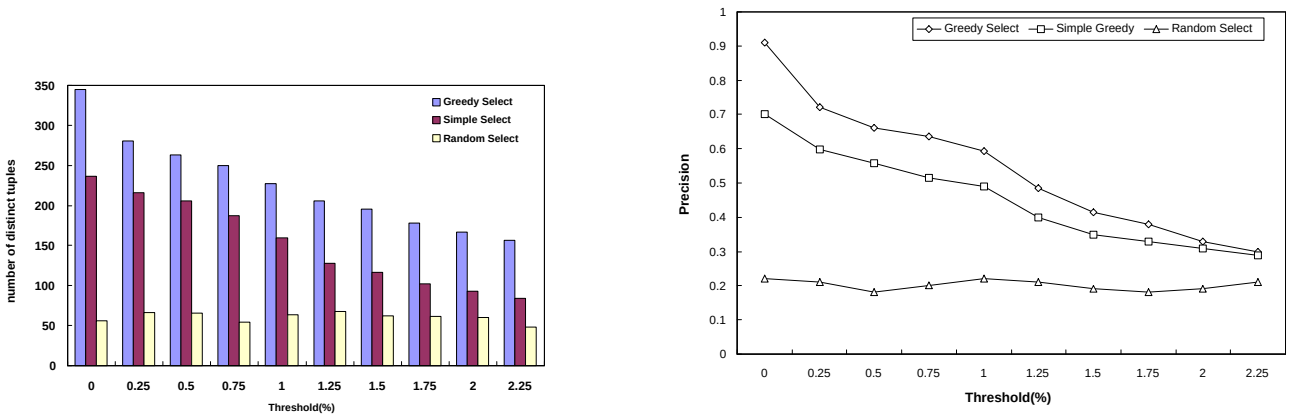
alogrithm and its usage by *StatMiner* can be found in [16, 17].



(a) Architecture of *StatMiner* (b) AV Hierarchies

Figure 2: Architecture of *StatMiner*

Evaluating the effectiveness of Statistics Mining: To demonstrate the effectiveness of the statistics learned by *StatMiner* and to see how the accuracy varies as we regulate the amount of statistics learned using the large class threshold, we designed a sample mediator system with 30 autonomous Web sources. The mediator uses the **Simple Greedy** and **Greedy Select** algorithms described in [7] to generate query plans using the source coverage and overlap statistics learned by *StatMiner*.



(a) Comparing Plan Coverages (b) Comparison of Precision

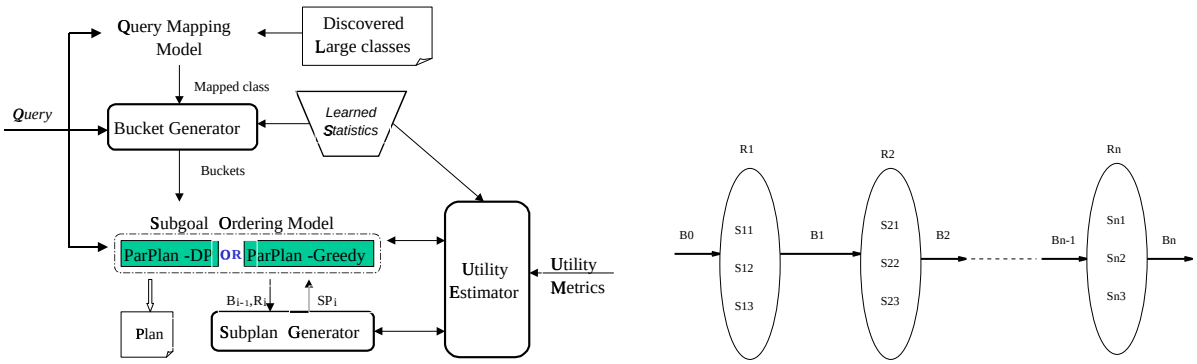
Figure 3: Comparison of Coverage and Precision

After probing the data sources and learning the coverage statistics in terms of association rules, we used a set of test queries to estimate the accuracy of learned statistics. The mediator maps any query to the lowest abstract class for which coverage statistics have been learned and generates plans using the three approaches: Random Select, Simple Greedy and Greedy Select. *Simple greedy* plans by greedily selecting

top k sources ranked according to their coverages, while *Greedy select* selects sources with high residual coverages calculated using both the coverage and overlap statistics. *Random Select* picks any k sources and does not depend on the available statistics. As can be seen from Figure 3(a) for all threshold values Greedy Select gives the best plan, while Simple Greedy is close second, but the Random Select performs poorly. The results demonstrate that the coverage as well as overlap statistics help significantly in ordering the source calls. In Figure 3(b) we compare the precision of plans generated for top 5 sources, as the large class threshold τ_c is varied, regulating the amount of learned statistics. We define the precision of a plan as the fraction of sources in the plan, which turn out to be the actual top k sources after we execute the query. We see that Greedy Select generates plans with highest precision for all values of class threshold τ_c. The experiments thus demonstrate that *StatMiner* is able to systematically trade accuracy for the time and space required to learn statistics. Additional details about the experimental setup and *StatMiner*'s efficiency in learning statistics measured in terms of time and space utilization are available in [16, 17].

3 Joint Optimization of Cost and Coverage with *Multi-R*

As a first step towards the realization of query optimization that is sensitive to multi-objective optimality metrics, we have developed and implemented a version of *Multi-R* that handles optimality metrics specified in terms of cost and coverage requirements. As we argue in [15], joint optimization is needed, as phased optimization techniques that consider optimizing coverage first and then cost, can lead to both plans and planning times that are considerably inferior. Figure 4(a) shows a schematic of the current implementation of *Multi-R*. *Multi-R* searches in the space of *parallel* query plans (see Figure 4(b)) which are essentially a sequence of source sets. Each source set corresponds to a subgoal of the query, such that the sources in that set export that subgoal(relation). The partial (parallel) plans are evaluated in terms of a general *utility* metric, that takes both cost and coverage of the plan into account. The cost and coverage estimates are made from the statistics that are learned by *StatMiner*.



(a) Architecture of *Multi-R* (b) Parallel Plans

Figure 4: Architecture of *Multi-R*

Cost and Coverage of Parallel Plans: A parallel query plan p (see Figure 4(b)) has the form

$$p = \big((...(sp_1 \bowtie sp_2) \bowtie ...) \bowtie sp_{n-1}\big) \bowtie sp_n, \text{ where } sp_i = (S_{i1} \cup S_{i2} \cup ... \cup S_{im_i}).$$

Here sp_i is a subplan and S_{ij} is a source relation corresponding to the i^{th} subgoal of the subgoal order used in the query plan. The subplan sp_i queries its sources in parallel and unions the results returned from the sources. The execution semantics of the plan p are that it joins the results of the successive subplans to

answer the query. We have developed cost models to compute the cost and coverage of parallel plans using the statistics gathered by *StatMiner*. Specifically we compute the cost (response time) of the parallel plans in terms of the source latencies and selectivity statistics, and coverage in terms of the source coverages and overlaps for large mediator classes as learned by *StatMiner*. Details can be found in the appendix or in [15].

Combining Cost and Coverage into a single Utility Metric: The main difficulty in combining the cost and the coverage of a plan into a utility measure is that, as the length of a plan (in terms of the number of subgoals covered) increases, the cost of the plan increases additively, while the coverage of the plan decreases multiplicatively. In order to make these parameters combine well, we take the sum of the logarithm of the coverage component and the negative of the cost component:[3]

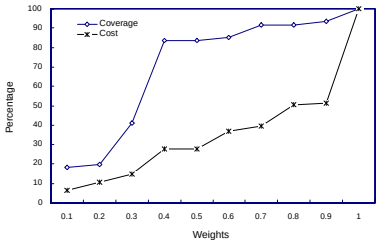
$$utility(p) = w * log(coverage(p)) - (1 - w) * cost(p)$$

The logarithm ensures that the coverage contribution of a set of subgoals to the utility factor will be additive. The user can vary w from 0 to 1 to change the relative weight given to cost and coverage.

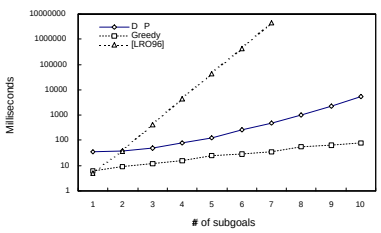
Generating query plans: Currently, *Multi-R* aims to find the single parallel plan with the highest utility. Our basic plan of attack involves considering different feasible subgoal orderings of the query, and for each ordering, generating a parallel plan that has the highest utility. To this end, we first consider the issue of generating the best plan for a given subgoal ordering. Given the semantics of parallel plans (see Figure 4), this involves finding the best "subplan" for a subgoal relation under the binding relation established by the current partial plan. *Multi-R* uses a greedy algorithm called *CreateSubplan* [15], to compute the best subplan for a subgoal R, given the statistics about the m sources $S_1, S_2, ..., S_m$ that export R, and the binding relation at the end of the current (partial) plan. *Multi-R* uses a greedy algorithm called *CreateSubplan* [15], to compute the best subplan for a subgoal R, given the statistics about the m sources $S_1, S_2, ..., S_m$ that export R, and the binding relation at the end of the current (partial) plan. *CreateSubplan* first computes the utility of all the sources in the bucket, and then sorts the sources according to their utility value. Next the algorithm adds the sources from the sorted bucket to the subplan one by one, until the utility of the current subplan becomes less than the utility of the previous subplan. Although the algorithm has a greedy flavor, the subplans generated by this algorithm can be shown to be optimal if the sources are conditionally independent [7] (i.e., the presence of an object in one source does not change the probability that the object belongs to another source). Under this assumption, the ranking of the sources according to their coverage and cost will not change after we execute some selected sources. The algorithm has $O(m^2)$ complexity. Once the *CreateSubplan* algorithm is in place, finding the parallel plan with the best utility involves sorting through the potential subgoal orders. In [15], we describe *ParPlan-DP*, a dynamic programming algorithm for this purpose. Currently, *Multi-R* also provides the *ParPlan-Greedy* algorithm for greedily choosing the subgoal whose best subplan can maximally increase the utility. A hill-climbing approach can be used to impove the utility of the plan. Further details and a complexity analysis can be found in [15].

Evaluating the Effectiveness of *Multi-R*: We have conducted a series of preliminary simulation studies to assess the effectiveness of the joint optimization approach used in *Multi-R* by comparing to the query optimization approach used in Information Manifold (IM) [13]. The experiments were done with a set of simulated sources, specified solely in terms of the source statistics. We used 206 artificial data sources and

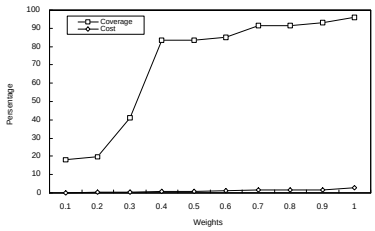
[3] We adapt this idea from [2] for combining the cost and quality of Multimedia database query plans, where the cost also increases additively and the quality (such as precision and recall) decreases multiplicatively when the number of predicates increases.



(a) Cost-Coverage tradeoffs offered by *ParPlan-DP* for different weights in utility function



(b) Planning time vs. query size



(c) Quality of solutions as compared to IM

Figure 5: Simulation studies comparing the effectiveness of *Multi-R* to the query optimizer used in Information Manifold [LRO96]

10 mediated relations covering all these sources [15]. Initially we tested the ability of our algorithms to handle a variety of utility functions and generate plans optimized in terms of cost,coverage or a combination of both. Figure 5[a] shows how the coverage and the cost of plans given by our *ParPlan-DP* changes when the weight of the coverage in the utility function is increased. We observe that as expected both the coverage and the cost increase when we try to get higher coverage. Also see that for a particular query at hand, there is a large area over which the coverage of a plan increases faster than the cost of a plan. The plots in Figure 5[b] compare the planning time for our algorithms with the phased optimization approach used in Information Manifold (IM) [13]. Both our algorithms incur significantly lower plan generation costs than the decoupled approach used in [13]. Also note that *ParPlan-Greedy* scales much better than *ParPlan-DP* as expected. In Figure 5[b], we plot the estimates of the cost and coverage of plans generated by *ParPlan-DP* as a percentage of the corresponding values of the plans given by the algorithm in [LRO96]. The cost and coverage values for the plans generated by each of the approaches are estimated from the source statistics, using the cost models we developed. The algorithms in [13] do not take account of the relative weighting between cost and coverage. So the cost and coverage of the plans produced by this approach remains the same for all values of w. The best plan returned by our algorithm has pretty high estimated coverage (over 80% of the coverage for w over 0.4) while incurring cost that is below 2% of that incurred by [13]. Even though our algorithm seems to offer only 20% of the coverage offered by [13] at w=0.1, this makes sense given that at w=0.1, the user is giving 9 times more weight to cost than coverage (and the approach of [13] is basically ignoring this user preference and attempting full coverage).

4 Summary and Current Directions

In this paper we motivated the need for multi-objective query optimization, and statistics gathering techniques for web-based data integration. We then described *Havasu*, a Web data integration system that is intended to support user queries with multiple, quite often conflicting objectives on autonomous Web sources that do not provide any statistical information. We described the design of *StatMiner* and *Multi-R*, two main components of *Havasu* that have been implemented, and presented results of their evaluation. We are currently making several extensions to *Havasu.* For *Multi-R* we are developing approaches for handling execution flexibility metrics as part of the joint optimization. Here we plan to consider both query plan metrics such as time to first tuple, as well as output rate characterizations of the plans (c.f. [23]). In the case of *StatMiner* we are developing approaches that are sensitive to the actual distribution of queries

encountered by the mediator. The idea is to make the resolution and size of statistics to be proportional to the frequency with which specific parts of the data are accessed (there by reducing the statistics for infrequently accessed parts of the data). Finally, for *Indra*, we are also developing techniques for monitoring the various quality metrics of the query plan [22], including coverage, in order to detect deviations from expected execution, as well as replanning techniques (c.f [11]) to handle the detected exceptions.

References

[1] S. Adali, K. Candan, Y. Papakonstantinou, and V.S. Subrahmanian. Query caching and optimization in distributed mediator systems. *Proceedings of SIGMOD.*, 1996.

[2] K.S. Candan. Query optimization in multi-media and web databases. Technical Report ASU CSE TR 01-003, Arizona State University, 2001.

[3] P. Dasgupta, P.P Chakrabarti, and S.C. DeSarkar. *Multiobjective Heuristic Search*. Friedr. Vieweg and Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1999.

[4] A. Doan and A. Levy. Efficiently ordering plans for data integration. *In IJCAI Workshop on Intelligent Information Integration, Stockholm, Sweden.*, 1999.

[5] O.M. Duschka, M.R. Genesereth, and A.Y. Levy. Recursive query plans for data integration. *Journal of Logic Programming.*, 43(1):49–73, 2000.

[6] D. Florescu, I. Manolescu A. Levy, and D. Suciu. Query optimization in the presence of limited access patterns. *Proceedings of SIGMOD.*, 1999.

[7] D. Florescu, D. Koller, and A. Levy. Using probabilistic information in data integration. *Proceeding of VLDB.*, 1997.

[8] Daniela Florescu, Daphne Koller, Alon Y. Levy, and Avi Pfeffer. Using probabilistic information in data integration. In *Proceedings of VLDB-97*, 1997.

[9] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[10] Z. Ives, A. Levy, D. Weld, D. Florescu, and M. Friedman. Adaptive query processing for internet applications. *Data Engineering Bulletin*, 23(2), 2000.

[11] N. Kabra and J. DeWitt. Efficient mid-query reoptimization. In *Proc. SIGMOD*, 1998.

[12] E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing recursive information gathering plans. *Proceeding of IJCAI.*, 1999.

[13] A.Y. Levy, A. Rajaraman, and J.J Ordille. Querying heterogeneous information sources using source descriptions. *Proceedings of VLDB, Bombay, India.*, pages 251–262, 1996.

[14] F. Naumann, U. Lesser, and J. Freytag. Quality-driven integratio of heterogeneous information systems. *Proceedings of VLDB*, 1999.

[15] Z. Nie and S. Kambhampati. Joint optimization of cost and coverage of query plans in data integration. *Proceedings of ACM CIKM, Atlanta, Georgia.*, 2001.

[16] Z. Nie, S. Kambhampati, U. Nambiar, and S. Vaddi. Mining source coverage statistics for data integration. *Proceedings of Workshop on Web Information and Data Management.*, 2001.

[17] Z. Nie, U. Nambiar, S. Vaddi, and S. Kambhampati. Mining coverage statistics for websource selection in a mediator. *ASU Technical Report*, 2002.

[18] T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems (2nd ed.).* Prentice Hall, 1999.

[19] C.H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *In proceedings of PODS*, 2001.

[20] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. *In proceedings of VLDB.*, 2000.

[21] A. Tomasic, L. Raschid, and P. Valduriez. A data model and query processing techniques for scaling access to distributed heterogeneous databases in Disco. *IEEE Transactions on Computers, special issue on Distributed Computing Systems*, 1997.

[22] T. Urhan and M. Franklin. Cost-based query scrambling for initial delays. *In proceedings of SIGMOD*, 1998.

[23] E. Viglas and J. Naughton. Rate-based query optimization for streaming information sources. *To appear in SIGMOD.*, 2002.

[24] R. Yerneni, C. Li, J. Ullman, and H. Garcia-Molina. Optimizing large join queries in mediation systems. *In proceedings of ICDE.*, 1999.

Appendix

A Cost Models for Parallel Query Plans

Estimating the Cost of a Parallel Plan: The execution costs are computed in terms of the response times offered by the various sources that make up the (parallel) plan. The response time of a source is proportional to the number of times that source is called, and the expected number of tuples transferred over each call. Since the sources have binding pattern limitations, and the set of feasible source calls depend on the set of call variables that can be bound, both the number of calls and the number of tuples transferred depend on the value of the *binding relation* preceding the source call. Specifically, suppose we have a plan p with the subplans $\{sp_1, sp_2, ..., sp_n\}$. The cost of p is given by: $cost(p) \doteq \sum_i responseTime(sp_i)$

The response time of each subplan $sp_i (= \{S_{i_1}, S_{i_2}, ..., S_{i_m}\} \in p)$ is computed in terms of the response times of the individual sources that make up the subplan. Since the sources are processed in parallel, the cumulative response time is computed as the sum of the maximum response time of all the sources in the subplan and a fraction of the total response time of the sources in the subplan:

$responseTime(sp_i) = max_{j \in [1,m]}\{responseTime(S_{i_j}, B_{i-1})\} + \beta \times \sum_{j \neq max_{RT}} responseTime(S_{i_j}, B_{i-1})$

where β is a weight factor between 0 and 1, which depends on the level of parallelism assumed to be supported by the system and the network. $\beta = 0$ means that the system allows full parallel execution of all the sources queries, while $\beta = 1$ means that all the source queries have to be executed strictly sequentially. Notice that the response time of each source is being computed in the context of the binding relation preceding that source call.

As explained in [15], for a source S under the binding relation B, we can compute $responseTime(S, B)$ in terms of the available source statistics. In doing this, we assume that the mediator uses a variation of semi-join based strategy [18] to evaluate the joins of the source relations (the variation involves handling the source access restrictions).

Estimating the Coverage of a Parallel Plan: For a query $Q : -R_1, R_2, ..., R_n$ and a plan $p = \{sp_1, sp_2, ..., sp_n\}$, the coverage of p, denoted by the probability $P(p|Q)$, $\frac{number\ of\ tuples(p)}{number\ of\ tuples(Q)}$. This will depend on the coverages of the subplans sp_i in p and the join selectivity factors of the subgoals and sources associated with these subplans. Let R_{sp_i} be the subgoal corresponding to the subplan sp_i=$\{S_{i_1}, S_{i_2}, ..., S_{i_m}\}$. We use $SF_J(B_{i-1}, sp_i)$ to denote the join selectivity factor between the sources within the i^{th} subplan and the binding relation resulting from joining the first $i-1$ subplans, and $SF_J(\widehat{B}_{i-1}, R_{sp_i})$ to denote the join selectivity factor between the i^{th} subgoal relation and the binding relation resulting from joining the first $i-1$ *subgoal relations*([15]). Coverage of the plan p can be estimated as:

$coverage(p) = \frac{\prod_{i=1}^{n}[card(sp_i) \times SF_J(B_{i-1}, sp_i)]}{\prod_{i=1}^{n}[card(R_{sp_i}) \times SF_J(\widehat{B}_{i-1}, R_{sp_i})]}$

If we assume that the subplans cover their respective relations uniformly (which is likely to be the case as the sizes of the subplans and their coverages increase), then we have $SF_J(B_{i-1}, sp_i) = SF_J(\widehat{B}_{i-1}, R_{sp_i})$. This, together with the fact that $\frac{card(sp_i)}{card(R_{sp_i})}$ is just the definition of $P(sp_i|R_{sp_i})$, simplifies the expression for coverage of p to $coverage(p) = \prod_{i=1}^{n}[P(sp_i|R_{sp_i})]$

The coverage of a subplan itself can be written in terms of the coverages provided by the individual sources exporting that relation:

$P(sp_i|R_{sp_i}) = P(\cup_{S_{i_j} \in sp_i} S_{i_j}|R_{sp_i})$

$$= P(S_{i_1}|R_{sp_i}) + P(S_{i_2} \wedge \neg S_{i_1}|R_{sp_i}) + ... + P(S_{i_m} \wedge \neg S_{i_1} \wedge ... \wedge \neg S_{i_{m-1}}|R_{sp_i})$$

The expressions of the form $P(S_{i_m} \wedge \neg S_{i_1} \wedge ... \wedge \neg S_{i_{m-1}}|R_{sp_i})$ refer to "residual coverages"–i.e., the residual additional coverage given by source S_{i_m} given that the other sources are going to be accessed anyway. Computing these residual coverage require the source overlap statistics from *StatMiner* (see [16] for details).

Combining Cost and Coverage into a single Utility Metric: The main difficulty in combining the cost and the coverage of a plan into a utility measure is that, as the length of a plan (in terms of the number of subgoals covered) increases, the cost of the plan increases additively, while the coverage of the plan decreases multiplicatively. In order to make these parameters combine well, we take the sum of the logarithm of the coverage component and the negative of the cost component:[4]

$utility(p) = w * log(coverage(p)) - (1 - w) * cost(p)$

The logarithm ensures that the coverage contribution of a set of subgoals to the utility factor will be additive. The user can vary w from 0 to 1 to change the relative weight given to cost and coverage.

[4]We adapt this idea from [2] for combining the cost and quality of Multimedia database query plans, where the cost also increases additively and the quality (such as precision and recall) decreases multiplicatively when the number of predicates increases.