Multiobjective Query Optimization

(Extended Abstract)

Christos H. Papadimitriou
Division of Computer Science
U. C. Berkeley
Berkeley, CA 94720
christos@cs.berkeley.edu

Mihalis Yannakakis
Bell Laboratories
Lucent Technologies
Murray Hill, NJ 07974
mihalis@research.bell-labs.com

Abstract

The optimization of queries in distributed database systems is known to be subject to delicate trade-offs. For example, the Mariposa database system allows users to specify a desired delay-cost tradeoff (that is, to supply a decreasing function $u(d)$, specifying how much the user is willing to pay in order to receive the query results within time d); Mariposa divides a query graph into horizontal "strides," analyzes each stride, and uses a greedy heuristic to find the "best" plan for all strides. We show that Mariposa's greedy heuristic can be arbitrarily far from the desired optimum. Applying a recent approach in multiobjective optimization algorithms to this problem, we show that the optimum cost-delay trade-off (Pareto) curve in Mariposa's framework can be approximated fast within any desired accuracy. We also present a polynomial algorithm for the general multiobjective query optimization problem, which approximates arbirarily well the optimum cost-delay tradeoff (without the restriction of Mariposa's heuristic stride subdivision).

1 Introduction

In Computer Science we have always been interested in *trade-offs* between various resources for the solution of computational problems; however, it was only very recently that trade-offs have begun to be considered as computational problems in their own right —probably the result of the advent of the Internet and the ensuing increasingly complex socio-economic context of computation.[1]

Consider the query optimization problem, for example, arguably the most important and complex problem in databases. Trade-offs between parallelism and communication in query optimization have been studied in [CHM, HM], and elsewhere. The Mariposa wide-area database system [SAP+] was architected to make such trade-offs explicit in an advantageous way. Mariposa assumes that a subquery can be executed in many diverse database sites, and each site submits a "bid" for the query, specifying a delay for delivering the result, and an associated cost. The query optimizer then compares these bids to a user-supplied cost-delay trade-off (a non-increasing function $u(d)$, specifying for each value d of the delay the amount of money the user is willing to pay in order to receive the query's results within time d), and attempts to determine the combination of subquery executions that maximizes the savings from this user-submitted curve. Mariposa's algorithm for this involves subdividing a query graph (obtained from a single-site optimizer) into horizontal "strides" that contain independent subqueries, compiling a cost-delay trade-off for each stride from the bids for these subqueries, and then combining the stride trade-offs by a greedy heuristic.

Expecting a user to submit a desired trade-off curve seems to us a little unrealistic. A much better user interface would be, instead, presenting *to* the user the costs and delays of several query plans, for his/her choice. Obviously, we only need to present query plans that are *undominated*, in that each has the property that no other plan is better in terms of both cost and delay. The set of all undominated solutions is called in the field of multiobjective (or multicriterion) optimization a *Pareto curve* —it captures the informal concept of a trade-off. Obviously, presenting to the user such a set of solutions to choose from is far superior to requiring a user-supplied trade-off and returning a solution that is in some rather arbitrary sense "most advantageous"

[1] At the recent FOCS conference, there was a session devoted almost exclusively to multi-objective optimization, the Internet, and the relationship between the two[FOCS].

with respect to the trade-off supplied, as Mariposa does.

By a fortunate coincidence, there has been recently much progress in the problem of computing Pareto curves. Even though there are often exponentially many solutions on a Pareto curve, and even the problem of finding the solution that optimizes one objective subject to the second being below a limit is NP-hard in very simple settings (e.g., shortest paths in a graph in which each edge has both a length and a delay), it is often possible to obtain in polynomial time an *approximate trade-off*, which is arbitrarily close to the true Pareto curve [PY].

In this paper we apply this approach in multiobjective optimization to the query optimization problem. First, we characterize the set of plans produced by Mariposa's greedy heuristic in terms of the Pareto curve, and show that for convex budget functions (a variant of) the heuristic finds the optimal solution (Theorem 1). However, we observe that in general the heuristic can lead us arbitrarily far from the best trade-off (Theorem 4). In contrast, we present an algorithm which, given the trade-offs from the strides as computed by Mariposa, constructs, for any ϵ, a set of query plans such that any other plan is approximately dominated (up to a factor ϵ) by a plan in the constructed set (Theorem 5). This approximate solution is called an *ϵ-Pareto curve.*

But these results concern the optimization problem as specified by the Mariposa optimizer, after a heuristic step in which the query has been partitioned into horizontal strides. It would be much more interesting if we could compute the approximate optimum trade-off of the given query graph *ab ovo*, essentially optimizing also the partition into strides (of course, finding the best query graph is a well-known hard problem, which is orthogonal to our interests here). We develop a polynomial approximation algorithm for this problem, based on dynamic programming (Theorem 7). Our algorithm also works in the more complicated case in which the computation can carried out in many sites, and delays and costs due to communication must be taken into account. Of course these algorithms are approximate; however, the approximation can become arbitrarily accurate at a reasonable computational cost; and the approximation is necessary, because the exact Pareto curves are both huge in size, and contain points that are hard to compute individually.

In Section 2 we describe the main relevant aspects of Mariposa; we also review the basic concepts and certain results from multi-objective optimization. In Section 3 we define the query multiobjective optimization problem in the context of Mariposa's query optimizer. We analyze the Mariposa heuristic in terms of the Pareto curve, and describe a polynomial algorithm for obtaining the ϵ-Pareto curve for this problem. Finally, in Section 4 we give the general algorithm for finding the ϵ-Pareto curve for a given query tree.

2 Background

2.1 Query Optimization in Mariposa

In Mariposa, a single-site optimizer first transforms an SQL query into a query graph. Then a *fragmenter* further subdivides the nodes by taking into account the way in which (fragments of) the base relations are stored in the various sites; the fragmenter also groups independent operations in the fragmented query graph into parallel strips called *strides.* Subqueries in a stride can be executed in parallel, but all subqueries of a stride must complete before the next stride can start (relaxing this to allow some pipelining is stated as a future problem in [SAP+]).

The atomic operations of the fragmented query plan are then passed to a *broker*, who invites various sites (assumed to belong to different corporate entities) to submit bids for each operation. A bid is a point in the cost-delay plane (plus an expiration date, a detail that we omit for simplicity). The broker combines the bids and produces a Pareto curve for the stride (that is, several plans, each with its own total cost and delay).

Last, a *coordinator* takes the Pareto curves for each stride, and attempts to combine them optimally to obtain the "optimum" execution plan. Among the possible execution plans, each with its cost c and delay d, Mariposa seeks the one that has the following property: If $u(d)$ is the amount the user would be willing (according to his/her trade-off curve submitted with the query) to pay for delay d, then the chosen query plan maximizes the difference $u(d) - c$. In other words, Mariposa's goal is to choose the point that maximizes the "vertical" distance from the user-supplied trade-off (this seems reasonable, yet rather arbitrary). This is the most intriguing and crucial algorithmic step in Mariposa's query optimizer.

Mariposa attempts to accomplish this optimization by a *greedy* algorithm: The coordinator first determines the plan of minimum delay: It is the combination of all plans of minimum delay for each stride; its delay d_0 is the sum of all minimum delays, and its cost c_0 is the sum of all associated (maximum) costs. Thereafter, the algorithm moves iteratively from one global plan to another by substituting one component of the current plan by another bid in the same stride, as follows. For each stride, and for each bid for the stride, the algorithm computes the cost gradient for the bid, i.e., the ratio of the cost savings over the additional delay incurred if we use this bid for the stride in place of the bid in the current global plan. The algorithm finds the the bid b among all bids of all strides that has the maximum cost gradient, and considers the new global plan obtained from the current plan by using bid b for the corresponding stride instead of the one in the current plan. If the new global plan is not better than the original (i.e.

does not yield a better profit $u(d) - c$) then the algorithm does not make the substitution and it terminates with the current plan. Otherwise, the algorithm makes the substitution, recomputes the cost gradients for the stride that contains the bid b that was substituted, and continues in the same manner with the new plan.

2.2 Multiobjective Optimization

Usually, an *optimization problem* is a way for associating with each instance x a set of feasible solutions $F(x)$, and, for each $s \in F(x)$, a cost $c(s)$. More formally, an optimization problem is a pair of polynomial algorithms f, c, such that, for any instance x (say, an SQL query) and candidate solution s (say, a query plan), $f(x,s)$ is a Boolean determining whether $s \in F(x)$ (i.e., it determines whether s is indeed a valid query plan for x), and $c(s)$ computes the cost of s (assumed to be a positive integer, e.g., an estimate of the number of operations needed to execute s).

A *k-objective optimization problem* is then a $(k+1)$-tuple of polynomial functions $f, c_1, \ldots, c_k$, where for each feasible solution s we have k different ways c_i of evaluating its cost. That is, each solution s has an associated k-vector of costs $c(s) =< c_1(s), \ldots, c_k(s) >$ (a point in k-dimensional space). We say that a k-vector $c =< c_1, \ldots, c_k >$ *dominates* another k-vector $c' =< c'_1, \ldots, c'_k >$ if $c_i \leq c'_i$ for all $i = 1, \ldots, k$ (with the inequality strict for some i). Similarly, we say that a solution s *dominates* another solution s' if $c(s)$ dominates $c(s')$. A feasible solution $s \in F(x)$, and its cost vector $c(s)$, is called *undominated* if there is no other solution s' that dominates it. Given an instance x of a multiobjective optimization problem, the *Pareto curve* of x, $P(x)$, is the set of all undominated cost vectors of solutions in $F(x)$. It is customary to use the term Pareto curve (in a slight abuse of notation) to refer also to a set of solutions that realize this set of undominated cost vectors. Intuitively, the points in the Pareto curve capture the notion of a "trade-off" between the various resources c_i.

For a typical multiobjective problem (even the shortest path problem with two objectives), the Pareto curve has an exponential number of points, and furthermore it is NP-hard to compute a particular desired point in it (say, the path with smallest first cost, and second cost bounded by B). Thus, multiobjective optimization seems computationally bleak. However, recently it was observed [PY] that all multiobjective optimization problems have a polynomially small *ϵ-approximate Pareto curve*: a set $P_\epsilon(x)$ of solutions that approximately dominate all other solutions up to a relative error of ϵ; that is, formally, for every solution $s \in F(x)$ there is a solution $s' \in P_\epsilon(x)$ such that $c_i(s') \leq (1+\epsilon)c_i(s)$ for all $i = 1, \ldots, k$. Thus, multiobjective optimization problems can be divided into those for which the ϵ-approximate Pareto curve can be computed in polynomial time for some reasonably small ϵ, and those for which it cannot. It is reasonable to consider the former to be "tractable."

Our results state that query optimization, starting from a specified query but with multiple choices for each base relation and operation varying with respect to cost and delay, is a tractable problem in this sense; the result holds both in the context of Mariposa's coordinator, and in general.

3 The Mariposa Coordinator Problem

Suppose that we have a query subdivided into n horizontal strides, and for each stride i we have a list of pairs $P_i = ((c_{i1}, d_{i1}), \ldots, (c_{im}, d_{im_i}))$ with $c_{ij} \geq c_{i,j+1}$ and $d_{ij} \leq d_{i,j+1}$ for all i and $j \neq m_i$; the pairs stand for the cost and delay of m_i possible query plans for each stride i. A feasible (global) query plan $(j_1, j_2, \ldots, j_n)$ is a choice of a plan j_i for each stride; its total cost is $\sum_i c_{i,j_i}$ and its delay is $\sum_i d_{i,j_i}$. This is a problem with two objectives, cost and delay. We call this the `Mariposa coordinator problem`.

We will first analyse Mariposa's greedy algorithm and relate it to the Pareto curve. Recall that the algorithm constructs a sequence of solutions $s_0, s_1, \ldots$, with associated set of points in the delay-cost plane $p_0 = (d(s_0), c(s_0)), p_1 = (d(s_1), c(s_1)), \ldots$. The algorithm makes sure that there is improvement in each step in the profit (i.e. that he quantity $u(d_i) - c_i$ is increasing in each step) and terminates when this is not the case. Let us ignore for now the user budget function and let the algorithm compute the complete sequence (this will be the case for example if u is a constant function.) If there are no ties among the bids in any steps, then this sequence is uniquely defined. If there are ties, then the set of solutions produced may depend on how the ties are broken. Let us assume for concreteness that when several bids from the same stride are tied, i.e. have the steepest descent, then the algorithm picks the one with the minimum cost; and ties among bids from different strides are broken arbitrarily.

For a multiobjective optimization problem, define its *convex Pareto curve* $CP(x)$ to be the Pareto curve of the convex hull of the set of cost vectors of the solutions, i.e. the lower left boundary of the convex hull. We identify $CP(x)$ with its set of vertices and the associated set of solutions. Thus, $CP(x)$ is a subset of the points (cost vectors) such that every other cost vector is dominated by a convex combination of some members of $CP(x)$.

In our case, the convex Pareto curve for each of the strides, as well as for the global solution set, is a polygonal line in the delay-cost plane. Consider the following algorithm.

Algorithm M'

1. For each stride i, compute the convex Pareto curve CP_i of the point set Pi. We consider the vertices as ordered in increasing delay (and decreasing cost), $CP_i = (p_{i0}, \dots p_{ir_i})$. Let s_0 be the global solution that consists of plan p_{i0} for each stride i; i.e., s_0 is a minimum delay solution.

2. Let s_t be the current global solution. If there is a stride i for which the corresponding component p_{ij} of s_t is not the last vertex of CP_i (equivalently, if $t < \sum_i r_i$), then choose that stride for which the next edge of CP_i has the steepest slope and replace p_{ij} by the next vertex $p_{i,j+1}$ of CP_i to form the new global solution s_{t+1}.

In case of ties in Step 2 between the strides (i.e. next edges with the same steepest slope), assume that Algorithm M' breaks them the same way as Mariposa.

Lemma 1 *Algorithm M' produces the same sequence of solutions as the Mariposa algorithm.*

Sketch of proof. By induction. The first solution is clearly the same. The induction step follows from the fact that for any stride i and vertex p_{ij} of CP_i, the plan of stride i that yields the steepest descent from p_{ij} is the next vertex of CP_i. □

One advantage of Algorithm M' is that it yields a more efficient implementation than the straightforward algorithm, namely the complexity is $O(n \log n)$ where n is the size of the input. Suppose that there are l strides, each with m plans (thus the input has size lm). Then the convex Pareto curve CP_i for each stride can be computed in time $O(m \log m)$. The sequence of solutions can then be generated in time $O(\log l)$ per iteration, using a priority queue to maintain the slopes of the next edges. Thus, the total time is $O(lm(\log l + \log m))$. The straightforward greedy algorithm would need an additional $O(m)$ time per iteration to recompute the gradients of the plans for the stride whose solution was modified; the total time would be $O(lm^2 + lm \log l)$.

Theorem 1 *The sequence of solutions computed by the Mariposa algorithm (and algorithm M') is the convex Pareto curve of the global solution set.*

Sketch of proof. Every vertex v of the convex Pareto curve CP is the (unique) optimal solution for some linear function of the cost and the delay $a.c + b.d$, with $a, b \geq 0$. Vertex v has the property that the slope $-b/a$ of the objective function lies between the slopes of the two edges of CP that are incident to v (or v is the first or last vertex of CP).

For linear objectives, the global optimization problem is trivial because we can optimize separately in each stride. That is, v consists of a solution v_i for each stride that optimizes $a.c + b.d$. Hence, v_i is a vertex of CP_i, all edges before v_i have slope that is steeper than the slope of the objective function, while the next edge does not. Since Algorithm M' chooses edges in order of slope, it follows that it will generate at some point the solution $v = (v_1, \dots, v_l)$; that is, it will never proceed in a stride i beyond the vertex v_i before all the other strides j reach the appropriate vertex v_j.

It follows that every vertex of the convex Pareto curve CP is produced by the algorithm. In case of ties in some steps, Algorithm M' (and Mariposa) may produce some additional intermediate points that lie in the middle of some edges of CP. In fact, the difference between the solution sequences generated by different ways of breaking the ties is in such intermediate points that get produced. □

One consequence is that, although the global solution set is exponential (if there are l strides with m plans each, then there are m^l solutions), the convex Pareto curve has a relatively small number of vertices - no more than the number of plans in the strides. (We note however that there may be an exponential number of solutions that correspond to intermediate points on the edges of the convex Pareto curve).

Corollary 2 *The convex Pareto curve has a linear number of vertices.*

A second consequence concerns the optimality of Mariposa's greedy algorithm for certain user budget functions. Recall that Mariposa assumes that the user provides a budget function $u(d)$ (a non-increasing function), and the algorithm aims to find a solution that maximizes the profit $u(d) - c$.

Corollary 3 *If the user function $u(d)$ is linear, then Mariposa computes the optimal solution. If $u(d)$ is convex, then the optimal solution occurs at a vertex of the convex Pareto curve, although the Mariposa algorithm may terminate prematurely before reaching the vertex.*

Sketch of proof. In the linear case, the claim follows from the proof of the Theorem above. (Note however that in this case optimization is trivial.) Suppose $u(d)$ is convex, i.e. $u(\lambda d_1 + (1-\lambda)(d_2)) \leq \lambda u(d_1) + (1-\lambda) u(d_2)$ for any d_1, d_2 and $0 < \lambda < 1$. A point $p = (d, c)$ that is not on the convex Pareto curve CP is dominated by a convex combination $p' = \lambda v_1 + (1 - \lambda)(v_2)$ of two vertices v_1, v_2 of CP. Then the profit of p is no greater than the profit of p' which in turn is no greater than the maximum of the profits of the vertices v_1, v_2. However, since the Mariposa algorithm stops when a step fails to improve the profit, it may terminate prematurely before reaching the optimal vertex. For example, suppose that $u(d) = 8 - 2d$ if $d \leq 2$, else 4, and the Pareto curve has three points (plans) with delay-cost (1,4), (2,3), (5,1). Then the algorithm will stop at the first plan for a profit of 2 (since the second point is worse), although the optimal plan is the third one for a profit of 4. □

Thus, for nonlinear budget functions, it is better not to terminate the algorithm if there is no immediate improvement, since a better point may be found later on (and the total number of generated global plans is in any case linear). For general (nonconvex) budget functions however even generating all the vertices of the convex Pareto curve may fail to produce an optimal solution. In fact the best vertex may be arbitrarily far from the optimum, or there may even not be a profitable vertex, although there are other plans with a positive profit.

Theorem 4 *For every number $M > 0$ there is a non-increasing function $u(d)$ and a set of Pareto curves for strides such that there is a way of combining strides with total cost C and delay D such that $u(D) - C > M \cdot [u(d_i) - c_i]$ for all the solutions (d_i, c_i) generated by the Mariposa algorithm .*

Sketch of proof. Suppose that there are three global plans with delay-cost $(1, 2M), (M+1, M+1), (2M+1, 1)$, and the user budget function is $u(d) = 2M + 1$ if $d \leq M + 1$, else $4M + 3 - 2d$. Then the optimal plan is the second one with profit M, while the other two plans have profit 1. However, $(M+1, M+1)$ does not belong to the convex Pareto curve as it is dominated by the average of the other two points. □

Thus, in general one needs to consider points of the Pareto curve that are not on the convex curve. Furthermore, a user interface that returns to the user the whole Pareto curve would be a major improvement over one that requires the user to supply a predefined trade-off curve.

How hard is it to compute the Pareto curve in this setting? It is not hard to see that this problem is intractable. First, it is easy to construct cases in which all exponentially many feasible solutions appear at the Pareto curve. Second, finding a single desired solution in the Pareto curve, say, the one that has cost less than B and is as fast as possible, given that constraint, is an NP-hard problem (easy reduction from the `knapsack` problem).

In view of these negative results, it is interesting to compute the ϵ-Pareto curve of this biobjective optimization problem —that is to say, the overall cost-delay tradeoff, approximated to within ϵ relative error.

Theorem 5 *There is an algorithm that computes the ϵ-Pareto curve of the* `Mariposa coordinator problem` *in time polynomial in the size of the input and $1/\epsilon$.*

Sketch of proof. One way of proving this is to realize that this problem can be reduced to the bicriterion shortest path problem (the problem facing a Web-based map service computing the distance/travel time trade-off of a trip). The reduction simulates each stride i by a set of m_i parallel edges, with costs and delays reflecting those of the options available at each stride. As the latter problem has a polynomial algorithm for finding its ϵ-Pareto curve [Han], the proof would be complete.

We also give here a dynamic programming algorithm, which will be generalized in the next section for general query trees. To start, we give a (pseudopolynomial) algorithm for finding the exact Pareto curve in time $O(nL)$, where L is the smaller of the maximum cost and the maximum delay of any point in the trade-off curve. Assume without loss of generality that the maximum delay is the smaller of the two. The dynamic programming algorithm computes in an array $A[1 \dots L]$, the minimum cost of a plan for each delay $d \leq L$. The computation is done one stride at a time. Having computed the information for the first $i-1$ strides we combine with the set of plans P_i of the ith stride: for $d = 1 \dots L$, $A(d) = min(A(d-1)$ do $min\{A(d-j) + c | (j, c) \in P_i, j < d\})$.

We can compute an ϵ-Pareto curve as follows. Partition the range of delays from the minimum delay up to L geometrically with ratio $1 + \delta = \sqrt{1+\epsilon}$, i.e. we define $O(\log_{1+\epsilon} L)$ bounds b_j with $b_{j+1} = b_j(1+\delta)$. Let $r = \lceil l/\delta \rceil$, where l is the number of strides. For each b_j, do the following. For each plan of each stride, transform its delay from d to $\lfloor dr/b_j \rfloor$. Compute the minimum cost global plan with transformed delay at most r; this can be done by the dynamic programming algorithm in time $nr = O(nl/\epsilon)$ (no need to compute array elements beyond r). Doing this for all b_j produces a set of global plans. Remove dominated plans from the set. The remaining ones form an ϵ-Pareto set. □

An approximate Pareto curve can be computed along similar lines also in the case where there is a set of sites, each subplan comes with an associated site that bids for it, and there are communication delays between the sites. The problem in this case can be still modeled as a bi-objective shortest path problem for a more complicated graph.

4 The Whole Hog

The query optimization problem solved in the previous section is only a subproblem of the Mariposa optimizer: It takes as given the partition into strides, for example. It would be more interesting if we could approximate a query's cost-delay trade-off in a more general setting.

Suppose that we are given a query tree T (the base relations could be fragments of the relations in the original SQL query, as in Mariposa), and, for each node i

(operation or input relations) we are given a list of pairs $P_i = ((c_{i1}, d_{i1}), \ldots, (c_{im_i}, d_{im_i}))$ standing for the cost and delay of various options for implementing this operation, given its input data. In the general distributed case, we assume that there is a set S of sites and every pair (c_{ij}, d_{ij}) of P_i (i.e. plan for execution of node i) has an associated site s_{ij} which is bidding to execute the node, and for each node i and pair of sites j, k there is a cost $c(i, j, k)$ and delay $d(i, j, k)$ for shipping the result of node i from site j to site k. It is assumed that a site can execute plans locally in parallel. We wish to choose an option for each node so as to minimize both total cost and total delay of the resulting query plan. We call this the `bicriterion query plan problem.` It is a generalization of the `Mariposa coordinator problem`, since that problem is the special case in which the query graph is a straight line (with the strides standing for meta-operations).

We observed in the last section that for some functions of cost and delay (eg. linear and convex), one can perform an exact global optimization, and in fact (a variant of) the Mariposa algorithm accomplishes this. This is no more the case, even for simple extensions (and without the generalization to many sites and communication costs and delays.)

Theorem 6 *The problem of finding a global solution that optimizes a given linear function of cost and delay for a given query tree is NP-hard.*

Sketch of proof. The query tree T consists of a path (as in the previous section) and an additional child of the root. The plans of the path simulate an instance of the knapsack problem, and the other child of the root has a single available plan. The objective linear function seeks to first minimize delay and then break ties by cost. $\square$

The problem is only weakly NP-hard; optimization in general can be solved in pseudopolynomial time as we will see below. As it turns out, a dynamic programming technique generalizing the one in the proof of Theorem 5 establishes that the more general bicriterion query plan problem can be efficiently approximated:

Theorem 7 *There is an algorithm that computes the ϵ-Pareto curve of the* `bicriterion query plan problem` *in time polynomial in the size of the input and $1/\epsilon$.*

Sketch of proof. We give first a pseudopolynomial algorithm. Let L be the delay of the minimum cost plan (this can be found in polynomial time). For each node i, site s and delay d, let $A(i, s, d)$ be the minimum cost of any plan that completes the computation of node i at site s within time d. These quantities can be computed bottom up by dynamic programming. For the leaves i use the given plans of P_i and the communication costs and delays. For an internal node i with set of children $C(i)$, for $d = 1 \ldots L$, compute $A(i, s, d)$ as the minimum of the following quantities: 1. $(A(i, s, d-1)$, 2. the minimum over all other sites j of $A(i, j, d - d(i, j, s)) + c(i, j, s)$, 3. the minimum over all plans (c_{ij}, d_{ij}) of P_i with site s of the quantities $c_{ij} + Sigma_{v \in C(i)}\{A(v, s, d - d_{ij})\}$.

To compute the ϵ-Pareto curve, we partition the range of delays geometrically as in Theorem 5, and then we scale and round the delays, and proceed in a similar manner as before. $\square$

5 Conclusion

We showed that the query optimization problem can be usefully studied within the scope of multiobjective optimization. It is interesting to recall here the work of [EHJ+]: We are given n web sources (horizontal fragments of a relation) each with its own delay, cost, and quality (roughly, the probability that a tuple will appear there). They study the tradeoff between cost (total), delay (maximum) and quality of the union of subsets of the fragments. It is shown in [PY] that this problem can be approximated in polynomial time. This problem can be viewed as the special case of a simple query, but with the additional twist that alternative sites (bids) are not equivalent but offer a range along a third dimension of quality. The more general query optimization problem could be studied also in a similar framework.

References

[CHM] C. Chekuri, W. Hasan, and R. Motwani. Scheduling Problems in Parallel Query Optimization. *Proc. of the Fourteenth ACM Symposium on Principles of Database Systems (PODS)*, 1995, pp. 255-265.

[CJK] T. C. E. Cheng, A. Janiak, and M. Y. Kovalyov. Bicriterion Single Machine Scheduling with Resource Dependent Processing Times. *SIAM J. Optimization*, 8(2), pp. 617–630, 1998.

[Cli] J. Climacao, Ed. *Multicriteria Analysis.* Springer-Verlag, 1997.

[ESZ] F. Ergun, R. Sinha, L. Zhang. An Improved FPTAS for Restricted Shortest Path. Submitted, 2000.

[EHJ+] O. Etzioni, S. Hanks, T. Jiang, R. M. Karp, O. Madari, and O. Waarts. Efficient Information Gathering on the Internet. *Proc. 37th IEEE Symp. on Foundations of Computer Science*, pp. 234–243, 1996.

[FOCS] *Proc. 41st IEEE Symp. on Foundations of Computer Science*, 2000.

[Han] P. Hansen. Bicriterion Path Problems. *Proc. 3rd Conf. Multiple Criteria Decision Making Theory and Application*, pp. 109–127, Springer Verlag LNEMS 177, 1979.

[Har] R. Hartley. Survey of Algorithms for Vector Optimization Problems. *Multiobjective Decision Making*, pp. 1–34, S. French, R.Hartley, L.C. Thomas, D. J. White Eds., Academic Press, 1983.

[HM] W. Hasan, R. Motwani. Optimization Algorithms for Exploiting the Parallelism-Communication Trade-off in Pipelined Parallelism. *Proc. of the 20th International Conference on Very Large Data Bases (VLDB)*, pp. 36-47, 1994.

[PY] C. H. Papadimitriou and M. Yannakakis. On the Approximability of Trade-offs and Optimal Access of Web Sources. *Proc. 41st IEEE Symp. on Foundations of Computer Science*, 2000.

[SAP+] M. Stonebraker, P. M. Aoki, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A Wide-Area Distributed Database System. *VLDB Journal*, 5:1, pp. 48-63, 1996.

[Wa] A. Warburton. Approximation of Pareto Optima in Multiple-Objective Shortest Path Problems. *Operations Research*, 35, pp. 70-79, 1987.