A Decision Theoretic Cost Model for Dynamic Plans

Richard L. Cole
Informix Software Inc.
485 Alberto Way
Los Gatos, CA 95032
rickcole@informix.com

Abstract

Since the classic optimization work in System R, query optimization has completely preceded query evaluation. Unfortunately, errors in cost model parameters such as selectivity estimation compromise the optimality of query evaluation plans optimized at compile time. The only promising remedy is to interleave strategy selection and data access using run-time-dynamic plans. Based on the principles of decision theory, our cost model enables careful query analysis and prepares alternative query evaluation plans at compile time, delaying relatively few, selected decisions until run time. In our prototype optimizer, these run-time decisions are based only on those materialized intermediate results for which materialization costs are expected to be less than the benefits from the improved decision quality.

1 Introduction

Query processing can be divided into query optimization and query execution; this dichotomy has been followed almost exclusively ever since the classic System R optimizer paper [SAC+79]. For queries with 10, 20, or 100 operators, this clear-cut distinction is no longer applicable. If each operator's selectivity is consistently underestimated by a mere 10%, the error after 10 operators is a factor of 2.8; the error after 20 operators is a factor of 8.2; and after 100 operators, it is a factor of 37,649. Beyond a level of query complexity somewhere between 10 and 20 operators, selectivity and cost estimates are more guesses than predictions, and plan choices are more gambles than strategic decisions [CVL+84, MCS88, IK91, Loh89]. Note that queries involving views, CASE tools, and data warehouse queries [KRRT98, Bri99] in particular often involve hundreds of operators. Thus, the problem addressed here is of great interest to database system vendors and implementors.

Not only estimates for intermediate result sizes, but also forecasts for available resources may err. The resource situation, e.g., available memory, processors, temporary disk storage, net bandwidth, internet site availability, and so on may change between compilation and execution, even from the time when a complex query is started to the time when the query completes its last operation. Since cost calculations and decisions among alternative plans depend on input sizes and available resources, selected optimization decisions and resource allocation decisions must be shifted from compile time into run time.

Research into this problem has taken several paths: improving the optimizer's statistical profile of the database system [MCS88, MRL98]; building static plans using a least-expected-cost metric [CHS99]; building start-up-time-dynamic plans [CG94]; materializing all intermediate results, i.e., Ingres Decomposition [WY76]; using intra-algorithm, i.e., record level adaptation [DG94, HRR99, AH00], decision theoretic techniques based on sampling [SBM93]; competitions between algorithms [Ant96]; heuristic approaches to materializing intermediate results [UFA98, BFMV00]; and partial plan re-optimization at run time [KD98, NWMN99, IFF+99]. Note that many of these techniques are compatible with each other.

In the industrial arena, only the Informix® Red Brick® Decision Server™[Bri99] delays preselected optimization decisions until run time. Red Brick applies information available at run time specifically to optimization decisions for star-schema queries [KRRT98, Bri99]. The server materializes the intermediate results of preliminary plans, each one composed from a star-schema *dimension table*, i.e., a primary key defining table referenced by a foreign key defining *fact table*. (Note that fact tables may contain billions of records.) Based on the materialized intermediate results, cost based, run-time decisions are made between compile-time generated alternative plans. Run-time decisions are also made between fact table orders, alternative join indexes, view maintenance algorithms, as well as the final determination of data partitioning functions and degrees of query parallelism.

In combination with join algorithms specialized for star-schema query processing, the run-time techniques employed by the Red Brick server enable efficient query evaluation of complex data warehouse queries involving many fact and dimension tables. One successful application's schema design involves 33 fact tables and 160 dimension tables. In this context, queries routinely join a many as 32 tables. Unfortunately, because of the difficulty in quantifying the advantages of these run-time techniques, their application is limited to star-schema query plans and sub-plans. What we need is a generalized compile-time approach to quantifying the advantages of such run-time decisions.

In this short paper, we briefly describe an extension of our previous work on start-up-time-dynamic plans [CG94], in order to address the problems posed by complex queries and uncertain cost model parameters. *Start-up-time-dynamic query evaluation plans* are dynamic plans implementing decisions typically made at compile time that are delayed until the start of a query's evaluation. *Run-time-dynamic query evaluation plans* are plans that make decisions between alternative, potentially optimal, algorithms, operator ordering, and plan shapes based upon additional knowledge obtained while evaluating the plan. For example, in our prototype run-time-dynamic plan optimizer [Col99], this additional knowledge is obtained by materializing intermediate results, thus providing exact cardinality and allowing accurate planning of subsequent query evaluation. Materializing intermediate results may also supply additional information about the distributions of attributes contained in the result, e.g., dynamically constructed histograms of join attribute values may facilitate improved costing of join algorithm alternatives [KD98].

The difficult aspect of this approach is determining which subsets of plan operators to materialize. Because interrupting the plan's data-flow and intermediate data materialization has an associated cost, the compile-time optimizer must trade off this cost against the benefit of additional knowledge and subsequently improved query evaluation. The techniques we employ to solve this problem are provided by statistical decision theory and it is the generalized application of decision theory to query optimization that is the key contribution of this work.

2 Cost Model

A cost model which optimizes run-time-dynamic plans must first capture the benefit of run-time decisions and, second, the cost of obtaining run-time information to make these decisions. Because we restrict ourselves to complete materialization of data, rather than partial sampling, we simplify the cost calculation of obtaining run-time information. It is simply the cost of the materialization itself, which is a function of the expected number and size of records to be read and written to a temporary result. In order to capture the benefit of run-time decisions we use the techniques of decision theory [RS68, Lin72].

2.1 Decision Theory

There are several elements involved in making a decision (largely from Raiffa and Schlaifer [RS68]): the set of possible actions, $A = \{a\}$, where a is an action chosen from the domain of actions A; the current state of the world $\Theta = \{\theta\}$, where θ is a particular state from domain Θ and choosing an action a depends a θ that may not be predicted with certainty; a set of possible experiments, $E = \{e\}$, that may improve our knowledge of Θ; the potential outcome of an experiment, $Z = \{z\}$; for every combination of a, θ, e, and z there is a consequence, often expressed as the utility, $u(\cdot,\cdot,\cdot,\cdot)$, defined on $A \times \Theta \times E \times Z$, that captures the benefit and cost of performing an experiment, observing an outcome, and taking a particular action; and we have prior estimates of the probabilities of the various states of the world before and after an experiment.

How do these principles apply to our problem? To optimize run-time-dynamic plans we apply them as follows: the domain A is the set of alternative query evaluation algorithms and plans; the state of the world, Θ, consists of known or estimated distributions for the parameters of the cost model, e.g., predicate selectivities or resources such as available memory; the set of possible experiments, E, is comprised of methods for obtaining better estimates for distributions on Θ that are only known approximately, e.g, sampling techniques, or as in our case, intermediate result materialization; the cost of the experiment, e, depends on the potential materialized outcome, z, since its cost is a function of the number of records materialized; in decision theory one typically wants to maximize the utility, $u(a, \theta, e, z)$, however, rather than defining utility as negative cost, $-c(a, \theta, e, z)$, we choose to minimize positive cost, $c(a, \theta, e, z)$, a perspective compatible with typical query optimizers; and we have prior probabilities for the uncertain parameters of the cost model, such as predicate selectivities or available resources, that capture their accuracy.

2.2 The Value of Information

Since the utility or cost of our experiments and actions are additive, we can think about the increase in the utility, or in our case decrease in optimizer cost, of an action for a given experiment. In decision theory, this increase is called the *value of information*. For a given experimental outcome we can compute the *conditional value of information*, but since we do not know these outcomes a priori we must resort to computing the *expected value of information*. Before running an experiment, e.g., before materializing intermediate results, we typically have some idea about the expected value of the cost model parameter, in this case intermediate result cardinality, and the accuracy with which we know the expected value. That is, we have a probability distribution for the expected value.[1] Using this previously known distribution, referred to as the *prior distribution*, we compute the *conditional cost with original information* (CCWOI), conditional with respect to a and θ, and the *expected cost with original information* (ECWOI) as follows.

The form $c(a|\theta)p(\theta)$ is the computation of CCWOI for cost that is a function of the action taken, a (the algorithm used), and the cardinality of the algorithm's input, θ, weighted by the probability density function representing our prior knowledge of the distribution of that cardinality, $p(\theta)$. Integrating their product over the range of θ (the state of the world), and minimizing over a (the algorithm set), gives the expected cost of making a particular algorithm choice, i.e., the ECWOI is $min_a \int_\Theta c(a|\theta)p(\theta)\,d\theta$.

Even without further development, the expected cost with original information is already a powerful decision theoretic concept when applied to a cost-based query optimizer. By minimizing the ECWOI, a query optimizer can build a static plan that has increased robustness without requiring start-up-time or run-time decisions. (This concept is similar to that explored by Chu et al.[CHS99]).

If we materialize intermediate query results, then for decisions between algorithms that directly depend on these results we will have perfect information for making decisions at run time. We can compute the *conditional cost with perfect information* (CCWPI), conditional on θ, and the *expected cost with perfect information* (ECWPI) as $min_a c(a|\theta)p(\theta)$ and $\int_\Theta min_a c(a|\theta)p(\theta)\,d\theta$ respectively. Note the relocation of the min_a term in these equations when compared to the ECWOI. From the ECWPI and ECWOI we may compute the *expected value of perfect information* or EVPI as ECWOI − ECWPI, providing an upper bound on the value of a perfect experiment.

[1] In lieu of continuous probability distributions we could instead use discrete distributions, as provided by histograms, etc.

2.3 The Value of Materialized Information

Materializing data may be an effective technique to obtain information for decision making immediately after materialization, in which case we have perfect information about cardinality and perhaps other information as well, e.g., domain histograms. However, the effectiveness of materializing data is reduced as subsequent operators operate on the materialized result. Decisions between alternative algorithms situated more than a single operator above (in the graph of operators) become less than perfect as intervening operators may introduce additional uncertainty, and the cost of materialization may ultimately outweigh its reduced benefit. The effect of this uncertainty is captured by the *expected cost with materialized information* (ECWMI).

To develop the ECWMI we use the *extensive* form of statistical analysis [Lin72]. Given the decision tree in

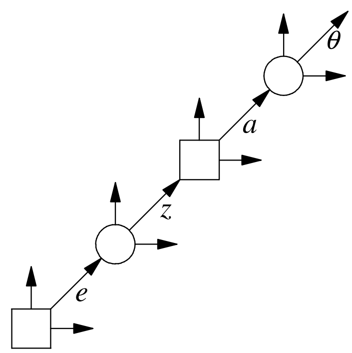


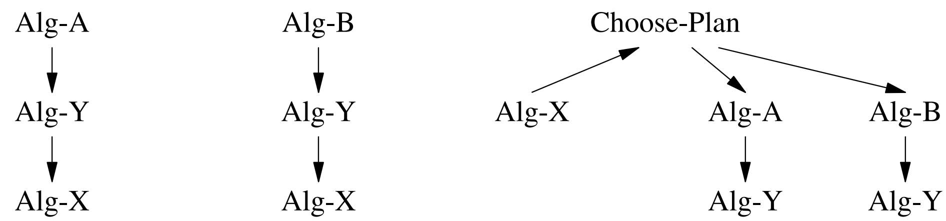
Figure 1: Experimental Decision Tree

Figure 1, where circles denote random variables and rectangles denote decisions, a formulation is developed to maximize the value of the experiment e. (Causality is in the direction of the arrows, from left to right and bottom to top.) First an experiment, e, is chosen, which produces experimental data, z. Based upon this information we choose an action, a, resulting in outcome θ and utility $u(a, \theta, e, z)$. Working backwards by first averaging the utility over θ where the applicable distribution is $p(\theta|z, e)$, it is then maximized over the set of available actions a, i.e., max_a, followed by averaging over the values of z, where the applicable distribution is $p(z|e)$, maximized over the set of experiments e, i.e., max_e, producing the equation $max_e \int_Z max_a \int_\Theta u(a, \theta, e, z)p(\theta|z, e)\, d\theta\, p(z|e)\, dz$, maximizing the expected utility.

The optimization of run-time-dynamic plans is analogous to the decision tree of Figure 1. The experiment e performed is the materialization of intermediate query results, the result of this experiment, z, is the size of the preliminary plan's intermediate result, the set of actions, a, are the set of alternative sub-plans available to Choose-Plan, and the final outcome, θ, is the actual result cardinality. Because we have chosen a priori to materialize intermediate results, the experiment, e, is fixed, i.e., we either materialize an intermediate result entirely or not at all. Therefore, we may dispense with the max_e term and references to e in general. A further simplification can be made because we know utility is not directly influenced by the experimental outcome. The cost of an algorithm in this example is only dependent on the cardinality of its immediate input. Therefore the ECWMI is $\int_Z min_a \int_\Theta c(a, \theta)p(\theta|z)\, d\theta\, p(z)\, dz$, and the *expected value of materialized information* (EVMI) is ECWOI − ECWMI.

2.4 An Extensive Form Example

Consider Figure 2 with static plans in diagrams (a) and (b), composed from generic algorithms Alg-A and Alg-B both having inputs Alg-Y and Alg-X, and in diagram (c) the run-time-dynamic plan having two alternatives, one of which will be chosen at run time based upon the cardinality of preliminary plan Alg-X materialized by the Choose-Plan operator as part of its decision procedure. If algorithm Alg-Y also introduces uncertainty in the input



(a) Static Plan A (b) Static Plan B (c) Run-Time-Dynamic Plan

Figure 2: Run-Time-Dynamic Plan with Additional Uncertainty

cardinality and uncertainty in the costs of algorithms Alg-A and Alg-B, then its effect must be considered in the design of the cost model.[2] This additional uncertainty is exactly what the ECWMI captures. In this example, the experiment performed is the materialization of the intermediate result produced by Alg-X, therefore we replace z in our ECWMI with x in order to associate it with Alg-X. Similarly, the final outcome is the intermediate result cardinality of Alg-Y and we substitute y for θ in our original definition of the ECWMI. With these modifications, the ECWMI of the example run-time-dynamic plan in Figure 2 (c) is $\int_X min_a \int_Y c(a, y)p(y|x)\, dy\, p(x)\, dx$, and after substituting for the cost of the specific alternative algorithms we have
$\int_X min[\int_Y c(\text{Alg-A}|y)p(y|x)\, dy, \int_Y c(\text{Alg-B}|y)p(y|x)\, dy]\, p(x)\, dx.$

2.5 A Prototype Plan and Preliminary Results

Figure 3 presents the graph of a dynamic plan produced by a prototype optimizer that applies the decision theoretic concepts just described [Col99]. Each node in the graph represents a typical operator [Gra93], with its name abbreviated as follows: EX - Execute (performs plan initialization), CP - Choose-Plan (evaluate preliminary plan(s) and choose from alternative plans) HJ - Hash-Join, FL - Filter (apply a predicate), FS - File-Scan (sequentially scan a file), FJ - Functional-Join (row identifier based join), and BS - Btree-Scan (with range restricting predicate). Each edge in a plan graph has an associated numeric label indicating the absolute order of inputs for an operator. For example, label 0 indicates the first input of an operator, label 1 the second, and so on. Preliminary plans, i.e., materialized sub-plans, are identified by dashed edges proceeding directly from the root node of a preliminary plan to the Choose-Plan operator(s) benefiting from its materialization.

The plan is for a four table join query having equijoin predicates and other, local predicates on all base tables. For one of these local predicates, the statistical profile of the database indicated much uncertainty in the predicate's selectivity. In this run-time-dynamic plan, there are many plan alternatives because of the decisions to be made between different base data algorithms, Functional-Join:Btree-Scan versus Filter:File-Scan, and different join orders for the Hash-Joins. Note that the operator that implements the uncertain predicate has been identified and its intermediate result is planned to be materialized. The materialized information is shown to be propagated to all Choose-Plan operators and determines all run-time choices. In the decision theoretic sense, the evaluation of this plan will be optimal.

Such qualitative results are encouraging, as are preliminary quantitative results. Large cost improvements have been realized for reasonable increases in compile-time optimization. For simple queries of a single table, experiments are easily constructed that produce relative cost improvements, by comparing the cost of dynamic plans to their analogous static plans, of more than a factor of three for very little additional optimization effort. More complex queries can produce similar improvements for an increase in optimization time that is reasonable depending on the context. For example, a query having 20 tables, 39 predicates, and 60 operators derived from

[2] Alg-Y need not be a single operator. It is representative of an arbitrarily complex sub-plan. Our decision analysis applies whether or not Alg-Y is a single operator or a sub-plan composed of multiple operators.

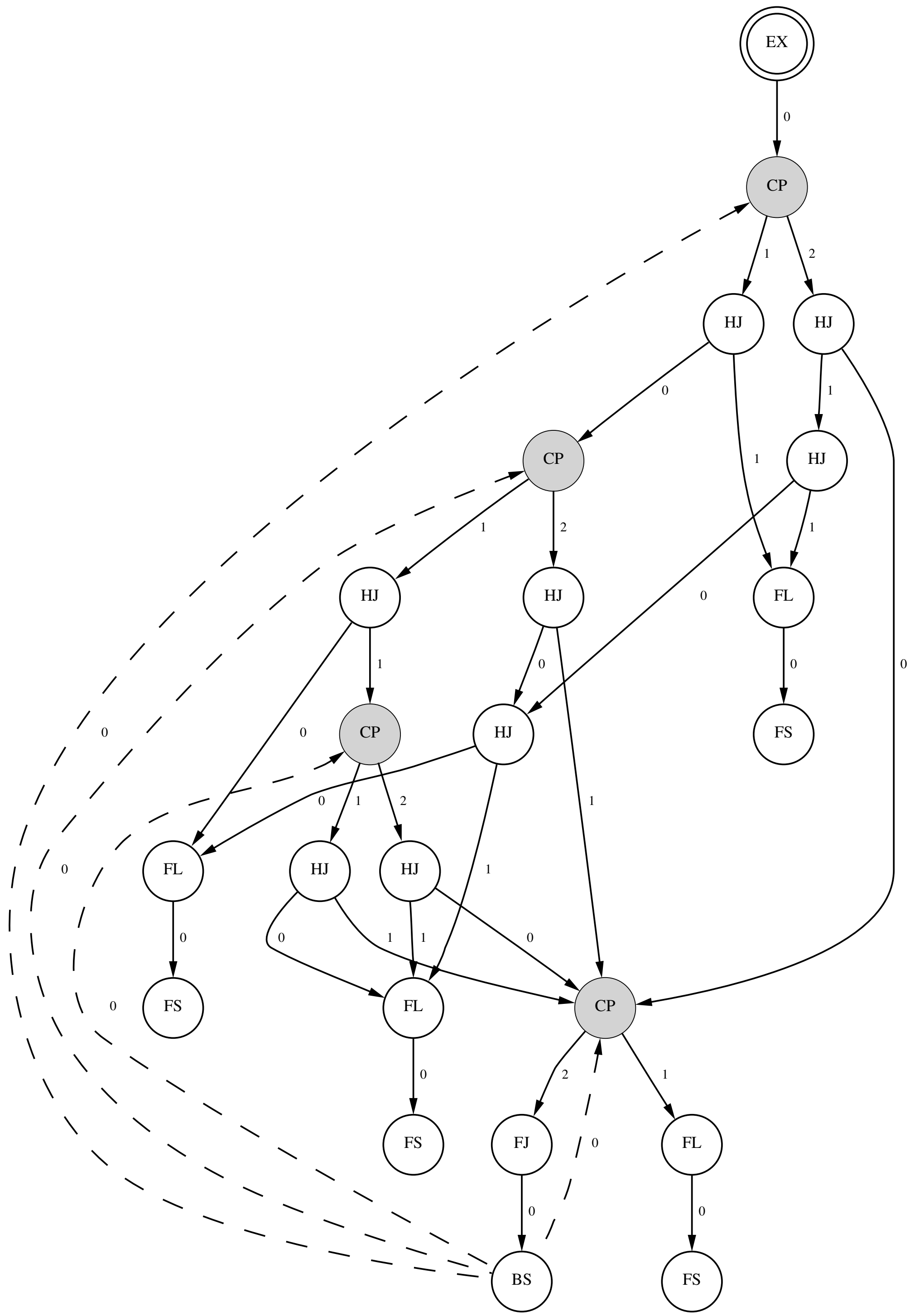


Figure 3: Discovering Localized Uncertainty: A Prototype Dynamic Plan

10^{21} equivalent physical forms was optimized in 20 seconds, compared to 1 second for the analogous static plan. In absolute terms, 20 seconds is an acceptable optimization time, especially when the result of that optimization is amortized over many query evaluations via precompiled plans.

3 Conclusion

The application of decision theory to query optimization was first considered by Seppi et al.[SBM93]. The focus of Seppi's work was on the application of Bayesian decision theory to three selected database query optimization problems: making decisions between alternative algorithms for evaluating simple selection predicates, join predicates over distributed relations, and the transitive closure of binary relations. The emphasis was on preposterior analysis, used to compute the optimal amount of sampling information to obtain before making a decision. (A later application of decision theory to query optimization was proposed by Chu et al.[CHS99]; however, the resulting plans remain static and there appears to be no adaptation during run time.)

Rather than focusing on any one specific query algorithm, we take a general approach that is independent of the algorithms under consideration, one that is useful for modeling the effective cost of inter-operator uncertainty. We also avoid dependency on any one specific distribution or family of data distributions, e.g., the exponential family of distributions as in Seppi's work. Finally, our emphasis is on the terminal analysis, i.e., analysis of the expected cost of algorithm decisions, rather than on the preposterior analysis, i.e., analysis of the expected effect of different sampling plans.

The accumulation of even small errors in the estimation of a predicate's selectivity can result in static query evaluation plans that are poorly optimized. Run-time-dynamic query evaluation plans are a promising approach to solving this problem of uncertain cost model parameters. To our knowledge, there has been no other work that proposes the generalized application of decision theory to query optimization for run-time adaptation.

References

[AH00] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *to appear Proc. ACM SIGMOD*, Dallas, TX, May 2000.

[Ant96] G. Antoshenkov. Query processing and optimization in Oracle RDB. *The VLDB Journal*, 5(4):229–237, January 1996.

[BFMV00] L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez. Dynamic query scheduling in data integration systems. In *Proc. IEEE Int'l. Conf. on Data Eng.*, pages 425–434, San Diego, CA, February 2000.

[Bri99] Red Brick. *Informix Red Brick Decision Server (Version 6.0): Administrator's Guide*. Informix Software, Inc., Los Gatos, CA, 1999.

[CG94] R. L. Cole and G. Graefe. Optimization of dynamic query execution plans. In *Proc. ACM SIGMOD*, Minneapolis, MN, May 1994.

[CHS99] F. Chu, J. Y. Halpern, and P. Seshadri. Least expected cost query optimization: An exercise in utility. In *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Sys.*, Philadelphia, Pennsylvania, May 1999.

[Col99] R. L. Cole. Optimizing dynamic query evaluation plans. *Ph.D. Thesis, University of Colorado at Boulder*, 1999.

[CVL+84] S. Christodoulakis, J. Vanderbroek, J. Li, T. Li, S. Wan, Y. Wang, M. Papa, and E. Bertino. Development of a multimedia information system for an office environment. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 261, Singapore, August 1984.

[DG94] D. L. Davison and G. Graefe. Memory-contention responsive hash joins. In *Proc. Int'l. Conf. on Very Large Databases*, page 379, Santiago de Chile, September 1994.

[Gra93] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[HRR99] J. M. Hellerstein, V. Raman, and B. Raman. Online dynamic reordering for interactive data processing. *Proc. Intl' Conf. on Very Large Data Bases*, September 1999.

[IFF+99] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *Proc. ACM SIGMOD*, pages 299–310, Philadelphia, Pennsylvania, May 1999.

[IK91] Y. E. Ioannidis and Y. C. Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *Proc. ACM SIGMOD*, page 168, Denver, CO, May 1991.

[KD98] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proc. ACM SIGMOD*, pages 106–117, Seattle, WA, June 1998.

[KRRT98] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*. John Wiley & Sons, Inc., August 1998.

[Lin72] D. V. Lindley. *Bayesian Statistics, A Review*. SIAM, Philadelphia, PA, 1972.

[Loh89] G. M. Lohman. Is query optimization a 'solved' problem? In G. Graefe, editor, *Proc. Workshop on Database Query Optimization*, page 13, Beaverton, OR, May 1989. Oregon Graduate Center CS TR 89-005.

[MCS88] M. V. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3):191, September 1988.

[MRL98] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proc. ACM SIGMOD*, pages 426–435, Seattle, WA, June 1998.

[NWMN99] K. W. Ng, Z. Wang, R. R. Muntz, and S. Nittel. Dynamic query re-optimization. *Proc. Conf. on Scientific and Statistical Database Management*, July 1999.

[RS68] H. Raiffa and R. Schlaifer. *Applied Statistical Decision Theory*. MIT Press, Cambridge, MA, 1968.

[SAC+79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proc. ACM SIGMOD*, page 23, Boston, MA, May-June 1979. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan Kaufmann, San Mateo, CA, 1988.

[SBM93] K. Seppi, J. Barnes, and C. Morris. A Bayesian approach to query optimization in large scale data bases. *ORSA J. of Computing*, Fall 1993.

[UFA98] T. Urhan, M. J. Franklin, and L. Amsaleg. Cost-based query scrambling for initial delays. In *Proc. ACM SIGMOD*, Seattle, WA, June 1998.

[WY76] E. Wong and K. Youssefi. Decomposition - A strategy for query processing. *ACM Trans. on Database Sys.*, 1(3):223, September 1976.