Robust Heuristics for Scalable Optimization of Complex SQL Queries

Gopal Chandra Das Jayant R. Haritsa
Database Systems Laboratory, SERC/CSA
Indian Institute of Science, Bangalore, INDIA

1 Introduction

Modern database systems incorporate a *query optimizer* to identify the most efficient "query execution plan" for executing the declarative SQL queries submitted by users. A dynamic-programming-based approach is used to exhaustively enumerate the combinatorially large search space of plan alternatives and, using a cost model, to identify the optimal choice. While dynamic programming **(DP)** works very well for moderately complex queries with up to around a dozen base relations, it usually fails to scale beyond this stage due to its inherent exponential space and time complexity. Therefore, DP becomes practically infeasible for complex queries with a large number of base relations, such as those found in current decision-support and enterprise management applications.

To address the above problem, a variety of approaches have been proposed in the literature. Some completely jettison the DP approach and resort to alternative techniques such as randomized algorithms (e.g. [7]) or genetic techniques (e.g. [5]), whereas others have retained DP by using heuristics to prune the search space to computationally manageable levels. In the latter class, a well-known strategy is "Iterative Dynamic Programming" **(IDP)** [3] wherein DP is employed bottom-up until it hits its feasibility limit, and then *iteratively* restarted with a *significantly reduced subset* of the execution plans currently under consideration. The experimental evaluation of IDP in [3] indicated that by appropriate choice of algorithmic parameters, it was possible to almost always obtain "good" (within a factor of twice of the optimal) plans, and in the few remaining cases, mostly "acceptable" (within an order of magnitude of the optimal) plans, and rarely, a "bad" plan.

While IDP is certainly an innovative and powerful approach, we have found that there are a variety of common query frameworks wherein it can fail to consistently produce good plans, let alone the optimal choice. This is especially so when *star* or *clique* components are present, increasing the complexity of the join graphs. Worse, this shortcoming is exacerbated when the number of relations participating in the query is scaled upwards.

Example. Consider the 15-relation "Star-Chain" join graph shown in Figure 1, where relation R_1 star-joins with relations R_2 through R_{11}, and R_{11} through R_{15} join in a chain formation – this join graph is structurally similar to Queries 8 and 9 of the TPC-H benchmark [8]. A hundred different instances of the Star-Chain join graph were implemented on the PostgreSQL engine [4], and optimized with DP and IDP (for a representative IDP parameter setting of $k = 7$, where k determines the number of DP levels executed in each iteration).

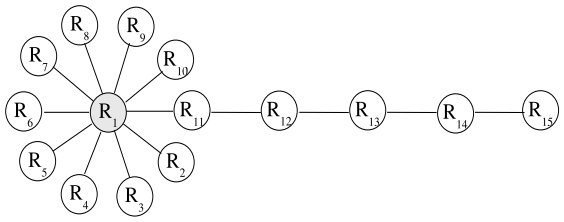

Figure 1. Star-Chain Join Graph

The relative performance results are shown in Table 1. Here, the classification of Good (G), Acceptable (A), and Bad (B) plans, is refined with the addition of Ideal (I), meaning the recommended plan is either identical to that produced by DP, or within 1% of this optimal. Additionally, the Worst-case (W) plan-cost increase ratio w.r.t. DP is given, and an overall plan-quality factor, ρ, defined as the Geometric Mean of the plan-costs normalized to the same metric w.r.t. DP, is tabulated.

Query Join Graph	Technique	Plan-Quality					
		I	G	A	B	W	ρ
Star-Chain-15	DP	100	0	0	0	1	1
	IDP	2	44	54	2	10.9	2.83
	SDP	80	20	0	0	1.2	1.02

Table 1. Plan Quality (DP, IDP, SDP)

The table shows that, relative to DP, for which all plans are Ideal by definition, a sizeable fraction of the plans deliv-

ered by IDP are *rather inefficient* – 56% are beyond a factor of 2 with regard to the optimal, and 2% are beyond a factor of 10. Further, IDP produces the ideal plan only for a very few (2%) queries. In the worst-case, the IDP plan is about 11 times slower than the optimal plan, and the ρ overall plan-quality metric is close to 3, way above the ideal value of 1.

Skyline Dynamic Programming

We have attempted to address the above problem of consistency in plan quality by proposing a new pruning strategy for the DP search space. Our heuristic, called "Skyline Dynamic Programming" **(SDP)**, is based on two novel premises: (a) Selectively applying pruning to only *local* segments of the join graph that are expected to be difficult to optimize, and not to the entire join graph; and, (b) Adopting a multi-way *skyline*-based pruning strategy on a sub-plan feature vector that incorporates *costs*, *cardinalities* and *selectivities*.

Through a detailed study, running to millions of complex star and star-chain queries on rich relational schemas implemented on the PostgreSQL engine, we have found SDP to be *robust* with regard to consistently providing high-quality plans – in fact, for a large fraction of the queries, it produces *ideal plans*. A quantitative instance is shown in Table 1, where SDP gives the ideal plan in 80% or more of the cases for Star-Chain-15, while the remaining sub-optimal choices are all good plans – in fact, very good plans, since in the worst-case, the plan selected by SDP is only 22% slower than the optimal. Finally, the ρ value is 1.02, very close to the ideal of 1.

Equally important, SDP's improvement is not achieved at the cost of increasing the optimization time and space overheads – on the contrary, due to its aggressive pruning strategy, SDP is completes the optimization process with overheads perceptibly lower than that of IDP. This is quantitatively shown in Table 2 where the space and time overheads of SDP are at least a third lower than that of IDP.

Query Join Graph	Technique	Memory (in MB)	Time (in sec)	Costing (in plans)
Star-Chain-15	DP	32.39	1.00	8.3E5
	IDP	7.39	0.20	1.3E5
	SDP	4.33	0.10	0.5E5

Table 2. Optimization Overheads

To put the above results in perspective, Figure 2 shows a plot of the plan-quality ρ against the the optimization overhead, for DP, IDP (with $k = 4$ and $k = 7$) and SDP. We see here that SDP produces a much better "knee-of-the-tradeoff" between input effort and output quality, as compared to IDP.

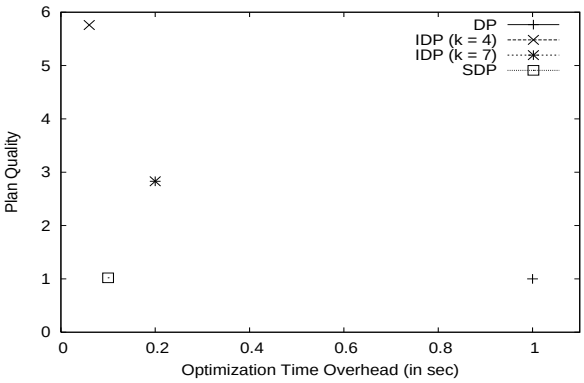

Figure 2. Plan Quality (ρ) vs. Effort Tradeoff

Effect of Scaling. When the Star-Chain join graph is scaled up to 23 relations, DP becomes computationally infeasible, due to running out of physical memory. However, both IDP and SDP are able to run to completion and in Table 3, we show IDP's performance relative to SDP, that is, treating SDP as ideal. We see there that the quality gap between SDP and IDP *increases*, with close to 90% of the IDP plans falling in the bad category relative to SDP. Further, with regard to the overheads, shown in Table 4, SDP requires about an *order of magnitude* less effort than IDP.

Query Join Graph	Technique	Plan-Quality I	G	A	B	W	ρ
Star-Chain-23	DP	*	*	*	*	*	*
	IDP	0	0	12	88	25.3	16.2
	SDP	100	0	0	0	1	1

Table 3. Scaled Join Graph: Plan Quality

Query Join Graph	Technique	Memory (in MB)	Time (in sec)	Costing (in plans)
Star-Chain-23	DP	*	*	*
	IDP	460.37	54.7	4.5E6
	SDP	55.33	1.08	0.4E6

Table 4. Scaled Join Graph: Overheads

In a nutshell, SDP consistently and efficiently produces high-quality query execution plans, as compared to prior pruning approaches. Moreover, like IDP, it can be easily integrated with current optimizers – in fact, as mentioned earlier, all our experiments have been conducted through *direct implementation* on the PostgreSQL engine.

2 The SDP Algorithm

A common characteristic of the previous approaches to limiting the DP search space was to apply the pruning universally over the *entire query join graph*. However, we have observed that in practice, it is the presence of *hub relations* (defined as relations that join with *three or more* relations) that are primarily responsible for the high overheads of DP in the optimization process. The notion of a hub relation applies not only to the base relations in the original query graph, but also to the intermediate *Join-Composite-Relations (JCRs)* that are computed during the optimization process. Based on this observation, SDP selectively applies pruning *only to JCRs containing hub relations*, leaving the remaining JCRs to be optimized under the aegis of the traditional exhaustive DP.

In its first iteration, SDP implements the standard DP algorithm, identifying the best access plan for each individual relation. Then, in the second iteration, all pair-wise join-composites (excluding cartesian products) of the base relations are enumerated, as in standard DP. These JCRs are split into two sets: **PruneGroup (PG)** and **FreeGroup (FG)**, with the splitting based on whether or not the JCR includes a complete hub from the immediately previous level – i.e. a "hub-parent". Subsequently the pruning strategy described below is applied, and the output is the set of length-2 "survivor JCRs". These survivor JCRs, along with all the survivor JCRs of previous levels then form the input to DP of the next level, and the process iteratively continues in this manner until a stage is reached where there are only two additional relations to be joined for each composite. At this point, by definition, there cannot be any hub-relations present, and therefore, the standard DP algorithm is employed for the last two levels.

Pruning Strategy. The pruning strategy in SDP has two steps: First, the JCRs in the PruneGroup are assigned to sub-groups that are formed with respect to the "root hubs", that is, the hub relations of the original join graph. The second step is to apply the function, described next, *within each sub-group*, to prune a subset of the JCRs present in the sub-group.

We characterize JCRs with a feature-vector comprised of the following attributes: [ROWS(R), COST(C), SELECTIVITY(S)], corresponding to the number of rows output by the JCR, the lowest cost of producing this output, and the output selectivity of the JCR relative to the product of the sizes of its base relations, respectively.

The *skyline* concept [1] is employed on this feature vector for pruning JCRs. Specifically, we compute a *disjunctive multiway skyline* on pairwise combinations of the RCS attributes in the feature vector. That is, we first identify the skyline set of JCRs based on their RC values, then the skyline set on the CS values, and finally the skyline set on the RS values. The JCRs featured in the three skylines are unioned, and all remaining JCRs are pruned. That is, we retain only those JCRs that are able to survive in at least *one of the three skylines*.

An example of the pruning process is shown in Table 5, where from the Prune Group on root hub 1, which consists of JCRs {1-2-3, 1-2-5, 1-3-5, 1-4-5, 1-5-6}, the survivor JCRs are {1-2-3, 1-2-5, 1-4-5, 1-5-6} while {1-3-5} is pruned (the digits are the relation identifiers).

Prune Group$_1$	Feature Vector [R,C,S]	*Skylines*		
		RC	*CS*	*RS*
1-2-3	[187638, 49386, 3.9E-5]	√	√	-
1-2-5	[122879, 52132, 1.0E-5]	√	√	√
1-3-5	[242620, 56021, 1.0E-5]	-	-	-
1-4-5	[241562, 55388, 6.65-6]	-	-	√
1-5-6	[385375, 52632, 4.5E-6]	-	√	√

Table 5. Multi-way Skyline Pruning

Further Details. The complete details of the design and implementation of the SDP algorithm, and its performance evaluation, are available in the full version of this paper [2].

References

[1] S. Borzsonyi, D. Kossmann and K. Stocker. *The Skyline Operator.* Proc. of 17th IEEE Intl. Conf. on Data Engineering (ICDE), 2001.

[2] G. Das and J. Haritsa. *Scalable Optimization of Complex SQL Queries.* Tech. Report TR-2006-01, DSL, Indian Inst. of Science, 2006. *http://dsl.serc.iisc.ernet.in/publications/report/TR/TR-2006-01.pdf*

[3] D. Kossmann and K. Stocker. *Iterative dynamic programming: a new class of query optimization algorithms.* ACM Trans. on Database Systems (TODS), 25(1), 2000.

[4] PostgreSQL Database System. *www.postgresql.com*.

[5] Postgres Genetic Optimizer. *www.postgresql.org/docs/7.4/static/geqo-intro2.html*

[6] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie and T. Price. *Access Path Selection in a Relational Database Management System.* Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1997.

[7] M. Steinbrunn, G. Moerkotte and A. Kemper. *Heuristic and Randomized Optimization for the Join Ordering Problem.* Intl. Journal on Very Large Data Bases (VLDB), 1997.

[8] Transaction Processing Performance Council. *http://tpc.org/*.