# Term paper: Randomized Algorithms and Heuristics for Join Ordering

Jim Wilenius
Computing Science Division
Dept. of Information Technology
Uppsala University
Box 337, SE-75105 Uppsala, Sweden
jim.wilenius@it.uu.se

May 18, 2007

## 1 Introduction

In the relational database setting today, large queries containing many joins are becoming increasingly common. In general the ordering of join-operations is quite sensitive and can have a devastatingly negative effect on the efficiency of the DBMS. Scheufele and Moerkotte proved that join-ordering is NP-complete in the general case [4]. For smaller queries however, less than approximately 10 joins, the optimal join strategy can be found by means of dynamic programming. However, the dynamic programming algorithm, proposed in [5], has a worst case running time of $O(2^N)$ (where $N$ is the number of joins), thus for queries with more than 10 joins, it becomes infeasible.

In the literature there are many alternative approaches to the join ordering problem, Steinbrunn et al [6] present a good overview. Approaches such as Iterative Improvement, Simulated Annealing, Genetic Algorithms, Two phase optimization and the probabilistic QuickPick etc all provide efficient alternatives although producing sub-optimal solutions to the join-ordering problem. In the next section a presentation to each of these will be made.

## 2 Local Search Heuristics and Randomized Algorithms

In order to provide acceptable join-orderings when dynamic programming is unavailable, heuristics and randomized algorithms can be used. Some examples are presented here.

## 2.1 The Join Ordering Problem

The join ordering problem can be viewed as two separate problems, on the one hand we have the problem of how to perform the actual join, on the other we have the problem of determining the order of performing the joins, it is the latter that is of interest here.

The search space can be regarded such that any feasible solution is a point in space, by searching we are trying to find the point associated with the lowest cost. The search is conducted by applying some set of rules to move from one state (point) to another in some approaches, and by random selection/generation in others.

## 2.2 Iterative Improvement (II)

The Iterative Improvement [7] is a variant of local search where the selection of which neighbour state to visit is performed at random. Neighbours are defined such that they can be reached by one *move* from the current state. In Iterative Improvement, two different type of move were defined and are applied with probability $\alpha$ and $(1 - \alpha)$ respectively. Given that the move results in a valid state, the moves are:

- Swap: Choose two relations at random and perform a swap.
- 3Cycle: Choose three relations at random $\langle a, b, c\rangle$, rotate them to the right $\langle c, a, b\rangle$.

Let $S$ and $minS$ be states and $cost(S)$ be the cost of state $S$. Let $neighbours(S)$ be the set of states reachable by one move from $S$: the Iterative Improvement algorithm is described in figure 2.2.

```
function IterativeImprovement
    while ¬stopping-condition do
        $S \leftarrow$ random state.
        while ¬local-optimum do
            $S' \leftarrow$ random state in $neighbours(S)$
            if $cost(S') < cost(S)$ then $S \leftarrow S'$
        end while
        if $cost(S) < cost(minS)$ then $minS \leftarrow S$
    end while
    return $minS$.
end function
```

Figure 1: Iterative Improvement (II)

Since it would be inefficient to check all the neighbours, a Local optimum is declared if in a large sample of the neighbourhood-set, no neighbour yields a better solution. Iterative Improvement could equally well be termed Iterated Local

Search as it fits very well in the Local Search group of heuristics. Other random-based heuristics mentioned in [7] such as Perturbation Walk (PW) and Quasi-random Sampling (QS) also fit very well into the definition of Local Search. QS selects new states completely at random which is the same as considering all states as neighbours. In Perturbation Walk, moves are selected at random, remembering the lowest cost state so far until a stopping criterion is met, this is a classical description of a Local Search heuristic.

I have not been able to find anywhere in the literature anything about the application of tabu-lists, this is a common technique that could potentially increase the efficiency/quality of the local search methods used.

## 2.3 Simulated Annealing

Simulated Annealing is based on the annealing process of crystals where, simply stated, first heating and then slowly cooling a liquid will result in crystals. It differs from the methods described so far in that it allows, with a certain probability, a non-improving move to be made. This probability is proportional to the *Temperature* of the system, which decreases over time. Ioannidis and Wong [3] implement and evaluate a Simulated Annealing system for optimization of recursive queries with satisfactory results. Other implementations can be found in the literature [7, 2, 6], see section 3.

## 2.4 Two Phase Optimization (2PO)

The Two Phase Optimization [2] heuristic was developed for select-project-join query optimization. It combines Iterated Improvement with Simulated Annealing. As can be deduced from it's name, 2PO consists of two phases, the first one is to run Iterative Improvement for a short period of time and using the resulting solution as the starting state for the Simulated Annealing. In the SA pass, a low initial temperature is used, limiting the search somewhat. The motivation for using the low temperature is that the cost function resembles a cup with uneven bottom. According to Ioannidis and Kang [2], the Two Phase Optimization outperforms the two original algorithms in both running time and quality, this is later verified in experiments conducted by Steinbrunn et al **??**.

## 2.5 Genetic Algorithms

Genetic Algorithms are based on Darwin's principles of survival of the fittest. By coding solutions as genomes and chromosomes and treating them as individuals in a population, individuals with good fitness (good solutions) procreate with an increased probability. By evolution-simulation better and better individuals are evolved.

Bennett and Ioannidis [1] applied genetic algorithms to query optimization. Compared to the System-R dynamic programming implementation in [5], they show good results for large queries, ($N = 16$ was the largest query tried).

Steinbrunn et. al [6] implement two different Genetic algorithms, one for left-deep trees and one for busy trees, see section 3 for a summary of their findings.

### 2.6 QuickPick

Waas and Pellenkoft [8] present a probabilistic bottom-up join order selection algorithm they named QuickPick. A normalized cost classification, the $\lambda$-Classification, is used to classify good or bad plans. From a large test set ($10^6$ plans) they conclude that the shape of the distribution of normalized costs resembles a gamma distribution. A $\lambda$ of less than 0.1 is considered a good plan.

Based on the favourable proportion of good to bad solutions, QuickPick works by building query plans bottom-up, adding randomly chosen edges and simultaneously calculating the partial cost as construction continues. If the partial cost of the current plan exceeds the cost of the best plan, it is discarded and a new permutation is generated. The procedure is repeated until a stopping criterion is met.

Their experiments evaluate QuickPick compared to Iterative Improvement and Uniform Selection. QuickPick very quickly finds good plans ($\lambda < 0.1$), after about 500 time steps, and its best plan ($\lambda$ near 0.03) after 1228 steps. Uniform Selection finds an equally good plan after 4400 steps whereas Iterative Improvement didn't find a solution with $\lambda < 0.15$ even after 5000 steps.

## 3 Discussion

Swami and Gupta [7] (1988), produce results showing that Iterative Improvement is superior to Simulated Annealing for non-recursive large join queries. Later, Ioannidis and Kang [2] (1990) show the opposite, that SA is almost always better than II for project-select-join queries. A more thorough investigation made by Steinbrunn et. al. in [6] confirm that SA outperforms II but they also give a more nuanced view. For left-deep trees they compare a Genetic Algorithm (GA), Simulated Annealing ($SA_H$ from [7]) and Iterative Improvement ($II_H$ from [7]). They show that $SA_H$ is superior to $II_H$ and that the GA is better than $SA_H$.

For bushy-trees they compare the 2PO, SA and II (SA and II referred to as $SA_{IO}$ and $II_{IO}$ taken from [2]) with GA (bushyGA). They conclude that for quality the 2PO is best but that $SA_{IO}$ and $II_{IO}$ are close ($SA_{IO}$ better than $II_{IO}$ most of the time).

A comparison in time was also made where they conclude that the bushyGA is the fastest followed by the GA, $SA_H$ and 2PO; the $II_{IO}$ and $SA_{IO}$ come last. The bushyGA solves 30-relation queries in about 50% of the time it takes the second place methods (GA, $SA_H$ and 2PO).

## References

[1] Kristin Bennett, Michael C. Ferris, and Yannis E. Ioannidis. A genetic algorithm for database query optimization. In Rick Belew and Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 400–407, San Mateo, CA, 1991. Morgan Kaufman.

[2] Y. E. Ioannidis and Younkyung Kang. Randomized algorithms for optimizing large join queries. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 312–321, New York, NY, USA, 1990. ACM Press.

[3] Yannis E. Ioannidis and Eugene Wong. Query optimization by simulated annealing. In *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 9–22, New York, NY, USA, 1987. ACM Press.

[4] Wolfgang Scheufele and Guido Moerkotte. On the complexity of generating optimal plans with cross products. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 238–248, 1997.

[5] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *SIGMOD Conference*, pages 23–34, 1979.

[6] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *The VLDB Journal*, 6(3):191–208, 1997.

[7] Arun N. Swami and Anoop Gupta. Optimization of large join queries. In *SIGMOD Conference*, pages 8–17, 1988.

[8] Florian Waas and J. Pellenkoft. Probabilistic bottom-up join order selection — breaking the curse of NP-completeness. In *157*, page 15. Centrum voor Wiskunde en Informatica (CWI), ISSN 1386-3681, 30 1999.