Dynamic and randomized Query Optimization algorithms to improve optimality of access plans

Ms. Deepali A. Patil, Ms. Aarti A. Patil, Prof. Thirumahal R.

Abstract— The goal of database performance tuning is to minimize the response time of your queries and to make the best use of your system's resources by minimizing network traffic, disk I/O, and CPU time. Query processing and optimization is a fundamental, if not critical, part of any DBMS. Queries, in a high level and declarative language e.g. SQL, which require several algebraic operations, could have several alternative compositions and ordering. Finding a "good" composition is the job of the optimizer. The primary goal of the query optimizer is to find the cheapest access path to minimize the total time to process the query. In this paper many different dynamic and randomized query algorithms that compute approximate solutions for producing optimal access plan are studied.

Keywords- DBMS**,** heuristic algorithm, join ordering, query optimization, query processing, randomized algorithm, SQL.

1 Introduction

While executing query in Relational database system, finding the optimal join ordering to create optimal access plan (query plan) is very difficult. Since SQL query is declarative (i.e never thinks that how the result obtained), there is a need to convert this declarative query to procedural query for finding effective plan of its execution. Join ordering is the primary focus of query optimization due to high processing cost. Traditionally, optimizations of expressions are done by traversing the complete solution space. But, it is applicable only where the 8-10 numbers of joins are used. This join ordering problem can be solved by three classes of algorithms. The first one focuses on the most important strategy, dynamic programming, which is the one used by essentially all commercial systems. The second one discusses a promising approach based on randomized algorithms, and the third one talks about other search strategies that have been proposed.

2 Deterministic Search Algorithm

This class of algorithm performs some sort of deterministic search [1] of solution space either through complete traversal or by applying some heuristics to prune the space.
It starts from base relations and build plans by adding one relation at each step. In dynamic programming it uses breadth-first strategy and builds all possible plans before choosing the "best" plan. In Greedy approach it uses depth-first strategy and build only one plan. Its disadvantage is that, for queries more than 10-15 joins, the running time explodes. The Heuristics [4] used to prune the search space are:

2.1 Selection Projection Heuristics

Selection and projection processing never generate transitional relations. Selection is processed upon first relation access and projections are applied at the time of generating output of other operations. This heuristics prunes only suboptimal solution. Separate processing of selection and projection would incur additional computational cost.

2.2 Cartesian product Heuristics

Relations are always combined through the joins not by Cartesian product.

2.3 Tree Form Heuristics

This is third heuristic that forms the execution plan trees where internal operand of every join is always a base relation and never a transitional result. Such a tree is called as left-deep tree. Traditionally, bushy trees were formed in which we find $(n^{2(n-1)} - 1)(n-1)!$ different solutions for n base relations. Now, the set of all possible left-deep access plans with n base relations is reduced to n!.

3 Genetic Algorithm

Genetic algorithms [1] make use of a randomized search strategy very similar to biological evolution in their search for good problem solutions. Although in this aspect genetic algorithms resemble randomized algorithms as discussed above, the approach shows enough differences to warrant a consideration of its own. The basic idea is to start with a random population and generate offspring by random crossover and mutation. The "fittest" members of the population (according to the cost function) survive the subsequent selection; the next gen-

- *Ms. Deepali A. Patil is a Lecturer in department of computer Engineering in SLRTCE,University of Mumbai, India, E-mail: deep.patil1987@gmail.com*
- *Ms. Aarti A. Patil is a Lecturer in department of computer Engineering in SLRTCE,University of Mumbai, India, E-mail: aarti.patil21@gmail.com*
- *Prof. Thirumahal R. is Assistant Professor in department of Information technology in TSEC, University of Mumbai, India, Email: r_thirumahal@yahoo.com*

eration is based on these. The algorithm terminates as soon as there is no further improvement or after a predetermined number of generations. The fittest member of the last population is the solution.
A problem with this approach might be that one member of the population is so prominent that it dominates the whole wheel. This way, it causes the disappearance of the other members' features.

4 Randomized algorithm

Randomized algorithms [4] view solutions as *points* in a solution space and connect these points by edges that are defined by a set of *moves*. It performs random walks in the state space. It moves from state to state with the goal of finding a state with the minimum cost. Two different moves are proposed for modifying these solutions: Swap and 3Cycle. Swap exchanges the positions of two arbitrary relations in the list, while 3Cycle performs a cyclic rotation of three arbitrary relations in the list. For instance, if R1R2R3R4R5 was a point in the solution space, the application of Swap might lead to R1R4R3R2R5, whereas 3Cycle could yield R5R2R1R4R3.
Two Randomized algorithms: Iterative improvement (II) and Simulated Annealing. This paper focuses on Iterative improvement (II):
.

4.1 Iterative Improvement (II):

The Iterative Improvement algorithm [4] starts at a random state. It then performs a number of downhill moves in order to find a local minimum. These moves are chosen as follows: Starting at a random state S, II explores the set of neighbors of S for possible moves. II determines the cost of S as well as that of a randomly chosen neighbor.

Algorithm:

function IterativeImprovementII

outputs minstate "*Optimized processing tree*"

Step 1: assign minimum state to infinity.

Step 2: pick a random state.

Step 3: while not at a local minimum reached,

- pick a random neighbor to the current state.
- if the neighbor has a lower cost, move there.

Step 4: if cost is lower than minimum cost then minimum state will be the state found at the above steps.

Step 5: repeat until time limit not exceeded.

Step 6: then return the lowest local minimum (i.e min state).

4.2 Simulated Annealing

Simulated annealing (SA)[5] is a random-search technique which exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system; it forms the basis of an optimisation technique for combinatorial and other problems.Simulated annealing [7] was developed in 1983 to deal with highly non-linear problems. SA approaches the global maximisation problem similarly to using a bouncing ball that can bounce over mountains from valley to valley. It begins at a high "temperature" which enables the ball to make very high bounces, which enables it to bounce over any mountain to access any valley, given enough bounces. As the temperature declines the ball cannot bounce so high, and it can also settle to become trapped in relatively small ranges of valleys. A generating distribution generates possible valleys or states to be explored. An acceptance distribution is also defined, which depends on the difference between the function value of the present generated valley to be explored and the last saved lowest valley. The acceptance distribution decides probabilistically whether to stay in a new lower valley or to bounce out of it. All the generating and acceptance distributions depend on the temperature. SA can find the global optimum.

Algorithm:

```
function SimulatedAnnealing
inputs state "Random starting point"
outputs minstate "Optimized processing tree"
minstate := state; cost := Cost(state); mincost := cost
temp := "Starting temperature"
do
  do
    newstate := "state after random move"
    newcost := Cost(newstate)
    if newcost <= cost then
     state := newstate
     cost := newcost
   else "With probability
        e^(newcost-cost/temp)"
    state := newstate
    cost := newcost
   end
   if cost < mincost then
     minstate := state
    mincost := cost
  end
  while "Equilibrium not reached"
        "Reduce Temperature"
while "Not frozen"
return minstate;
```

SA's major advantage over other methods is an ability to avoid becoming trapped in local minima. The algorithm employs a random search which not only accepts changes that decrease the objective function f (assuming a minimisation problem), but also some changes that increase it. T. The latter are accepted with a probability p = exp (-df / T) (1)

where df is the increase in f and T is a control parameter, which by analogy with the original application is known as the system ′′*temperature*" irrespective of the objective function involved.

5 Conclusion

To produce an optimal access plan and optimized join expressions deterministic, genetic and randomized algorithms are used. Because of new database applications, the complexity of the optimization task has increased; more relations participate in join expressions than in traditional relational database queries. Enumeration of all possible evaluation plans is no longer feasible. But in terms of running time randomized algorithm and genetic algorithms are better than deterministic algorithm because they have larger solution space. Algorithms that compute approximate solutions, namely heuristic, randomized and genetic algorithms, show different capabilities for solving the optimization task. Randomized and genetic algorithms are much better suited for join optimizations; although they require a longer running time, the results are far better.

References

[1] Prof. M.A.Pund, R.Jadhao, P.D.Thakare, "A Role of Query Optimizatio in Relational Database", *IJSER,Volume* 2,Issue 1,January-2011.

[2] A Swami, Optimization of Large join Queries Combining Heuristics and combinatorial Techniques, in Proceedings of the 1989 ACM-SIGMOD Conference, Portland, OR, June 1989.

[3] G. Antoshenkov, "Dynamic Query Optimization in RdblVMS", Proc. IEEE Int „1. Conf on Data Eng., Vienna, Austria, April 1993,538.

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

[5] Ingber, L., 1993, "Simulated annealing: practice versus theory", *Mathl. Comput. Modelling* **18**, 11,29-57.

[6] Yannis E.Ioannidis and Youngkyung Cha Kang: Randomized Algorithms for Optimizing Large Join Queries.

[7] Franco Busetti "Simulated annealing overview"