AN ADAPTIVE GENETIC ALGORITHM WITH DYNAMIC POPULATION SIZE FOR OPTIMIZING JOIN QUERIES

Stoyan Vellev

Abstract: The problem of finding the optimal join ordering executing a query to a relational database management system is a combinatorial optimization problem, which makes deterministic exhaustive solution search unacceptable for queries with a great number of joined relations. In this work an adaptive genetic algorithm with dynamic population size is proposed for optimizing large join queries. The performance of the algorithm is compared with that of several classical non-deterministic optimization algorithms. Experiments have been performed optimizing several random queries against a randomly generated data dictionary. The proposed adaptive genetic algorithm with probabilistic selection operator outperforms in a number of test runs the canonical genetic algorithm with Elitist selection as well as two common random search strategies and proves to be a viable alternative to existing non-deterministic optimization approaches.

Keywords: genetic algorithms, query optimization, join ordering, randomized algorithms

ACM Classification Keywords: H.2.4. Query processing, H.3.4. Performance evaluation (efficiency and effectiveness)

Conference: The paper is selected from International Conference "Intelligent Information and Engineering Systems" INFOS 2008, Varna, Bulgaria, June-July 2008

Introduction

Queries in a relational database management system (RDBMS) are defined in a declarative, non-procedural language, such as SQL. This raises the need to transform the declarative query into a procedural, effective plan for its execution. Each query can be mapped to a set of execution plans which are equivalent in terms of the result they generate but the execution cost of the different plans can vary by many orders. The execution plan is selected from the set of all alternatives by a dedicated RDBMS module – the Query Optimizer.

Due to the high processing cost, the evaluation of joins and their ordering are the primary focus of query optimization. Traditionally, the optimization of such expressions is done by complete traversal of the solution space (probably utilizing some pruning techniques). This is a possible approach for most of the classic database applications, where the size of the query (the number of joined relations) rarely exceeds 8-10, but it is completely inapplicable to some contemporary databases (Object-Oriented Databases, Multimedia Databases) and database applications such as Decision Support Systems (DSS), Online Analytical Processing (OLAP), Data Warehousing, Geographical Information Systems (GIS), etc. Queries in such applications may involve tens or even hundreds of joined relations.

This paper is focused on the optimization of a particular type of queries – single flat conjunctive queries, also known as selection-projection-join (SPJ) queries or non-recursive Horn clauses.

The Problem

Each query Q against a relational database defined by some data dictionary $\mathcal{D}$ with a set of relations $\mathcal{R}$ is represented by the ordered tuple (R^q, P^q), where $R^q = \{R_i \mid R_i \text{ is referenced in } Q\}$, $R^q \subseteq \mathcal{R}$ and $P^q = \{p_i(R^i_j, R^i_k) \mid p_i$ is a join predicate in Q, $R^i_j, R^i_k \in R^q\}$.

A *query execution plan* (QEP) of Q is a binary tree, in which the internal nodes represent join operator implementations (*join methods*), e.g. nested-loop join, merge join or hash join, and the leaves are base relations. Unlike the query itself (that has only declarative semantics), the query execution plan contains the procedural

information about how to obtain the query result. Each execution plan has a *cost* that reflects the computational resources needed to evaluate it.

The problem is, given a query *Q* to find the execution plan (from the set of all equivalents) with the lowest cost that evaluates it (the global optimum). Since the combinatorial explosion makes the exhaustive solution search impossible, the aim will be restrained to finding a good *local* optimum.

Given a query *Q* of *n* relations against a database supporting a set $\mathcal{S}$ of different join methods, there are $\frac{1}{n}\binom{2(n-1)}{n-1}$ possible QEP tree structures, for each of it its *n* leaves can be ordered in *n*! different ways and each internal node is selectable from *s* different join methods, where *s* = | $\mathcal{S}$ |. In this work we limit our considerations to the space of *left-recursive* solutions (containing all trees for which it holds that each of their nodes has a base relation for a right successor), but there are still $n!s^{n-1}$ different solutions.

The problem of finding the optimum join order can be assumed a static optimization problem – although the database state is dynamic (it may change during the optimization of a query), the cost function depends on database statistics (rather than on the real-time database state) which can be considered static as they are updated in a controlled way and do not interrupt any ongoing optimizations.

There are a number of polynomial-complexity algorithms for solving some special cases of the problem. All of them however impose some major restrictions on the form of the queries, the type of the cost function used, the particular implementation of the join methods, etc. The join ordering problem in its general form is unfortunately $\mathcal{NP}$-complete.

Related Work

Due to the inapplicability of deterministic optimization algorithms (different variations of the classical dynamic optimization with pruning) to the join ordering problem for large queries (where *large* is usually defined as queries with 8 or more joins), the problem has been approached by two classes of non-deterministic algorithms – randomized and genetic.

Two well-known randomized algorithms have been applied to the problem of optimizing large join queries – *Iterative Improvement* and *Simulated Annealing*, as well as a combination of the two [6]. Though the effectiveness of randomized algorithms strongly depends on the shape of the solution space, they generally prove to be a possible alternative to deterministic search for large queries.

Genetic algorithms (GAs) have been first applied to query optimization in [5] and [4]. The fitness function used requires backward transformation from chromosome to tree representation, which is complex and with high computational cost. The chosen crossover operators have a serious flaw – they disrupt the chromosome structure, transforming two valid parent chromosomes into an invalid one, which then needs to be “repaired” to become a correct solution encoding. Despite these shortcomings, the achieved results are promising. Later in [3] some of these disadvantages have been overcome.

A theoretical comparison of randomized and genetic optimization algorithms concluded that many GAs (the canonical GA in particular) are characterized by higher probability of finding good solutions than randomized algorithms, as long as the solution space fulfills several restrictions. These restrictions however are weak and hold for almost any choice of the genetic operators [2].

Currently the only popular non-experimental genetic SQL query optimizer is the GEQO (GEnetic Query Optimizer) in the Postgres (PostgreSQL) RDBMS. It considers only left-recursive solutions, implements an Elitist selection operator, a simple edge recombination crossover and does not apply mutation. The population size is fixed.

Recently, self-adaptation in genetic algorithms (population size adaptation in particular) is receiving great attention [1]. However, no adaptive genetic algorithm has been applied so far to database query optimization.

The Algorithm

We introduce an adaptive genetic algorithm with dynamic population size as an efficient solution to the optimal join ordering problem.

Solution representation (Coding). The coding operator Θ transforms an individual ξ_j into a vector of genes, each gene being an ordered tuple of a relation number and a join method number:

$\Theta(\xi_j) = \Theta(R_{i1} \bowtie_{p1} R_{i2} \bowtie_{p2} \ldots \bowtie_{pn-1} R_{in-1} \bowtie_{pn-1} R_{in}) \rightarrow ((i_1, p_1), (i_2, p_2), \ldots, (i_n, p_n))$

Mutation. The mutation operator M transforms an individual ξ_j into a new individual ξ'_i by swapping two randomly selected genes γ_x and γ_y and changing the join method of another randomly chosen gene γ_m:

$M(\xi_j) = M(((i_1, p_1), \ldots, (i_x, p_x), \ldots, (i_m, p_m), \ldots, (i_y, p_y), \ldots, (i_n, p_n))) \rightarrow (((i_1, p_1), \ldots, (i_y, p_y), \ldots, (i_m, p'_m), \ldots, (i_x, p_x), \ldots, (i_n, p_n)))$; $x, y, m \in \{1, 2, \ldots, n\}$, $x \neq y$, $p_m \neq p'_m$

The *mutation rate* μ is the probability of an individual ξ_j to mutate on each generation, i.e. $M(\xi_j) \equiv \xi_j$ with probability (1 - μ). The mutation operator is never applied to the individual with maximum fitness in the population.

Crossover. The crossover operator X combines the chromosomes of two generation-g individuals ξ^g_i and ξ^g_j to obtain two new generation-$(g + 1)$ individuals ξ^{g+1}_i and ξ^{g+1}_j. Random locus x is chosen, the two parent chromosomes are split at that locus and each of the two offspring receives a whole fragment from one of the parents (the first child - the left and the second child - the right relative to the locus) and the rest of the chromosome is filled up with the missing genes in the order they occur in the second parent. This guarantees both structural and functional similarity of the children with both their parents.

$X(\xi^g_i, \xi^g_j) = X(((i_1, p_1), \ldots, (i_x, p_x), (i_{x+1}, p_{x+1}), \ldots, (i_n, p_n)), ((j_1, q_1), \ldots, (j_x, q_x), (j_{x+1}, q_{x+1}), \ldots, (j_n, q_n))) \rightarrow \{((i_1, p_1), \ldots, (i_x, p_x), (j'_{x+1}, p'_{x+1}), \ldots, (j'_n, q'_n)), ((i'_1, p'_1), \ldots, (i'_x, p'_x), (j_{x+1}, q_{x+1}), \ldots, (j_n, q_n))\}$, $x \in \{1, 2, \ldots, n\}$

Each individual ξ_j in the population selects a crossover partner from its *neighborhood*, defined as its k neighbors by index in the population vector, $k = const$. The probability of an individual ξ_j from the neighborhood to be chosen for a mating partner is proportional to its fitness $\varphi(\xi_j)$.

Selection. The selection operator Σ transforms one population ξ into another population $\xi' \subseteq \xi$:

$\Sigma(\xi) = \Sigma(\{\xi_1, \xi_2, \ldots, \xi_k\}) = \{\xi_{j1}, \xi_{j2}, \ldots, \xi_{jm}\}$, $m \leq k$.

We propose an algorithm called *Probabilistic Selection with Adaptive Population Size*. All classical selection algorithms keep the population size fixed. This simplifies the algorithms but it is an artificial restriction and does not follow any analogy to biological evolution, where the number of individuals in a population varies continuously in time, increasing when there are high-fit individuals and abundant resources and decreasing otherwise. Intuition hints that it may be beneficial for the population to expand in the early generations when there is high phenotype diversity and there is opportunity to "experiment" with different characteristics of the individuals, and to shrink with the increase of population convergence, when the unification of the individuals in terms of structure and fitness no longer justifies the maintenance of a large population and the higher computational costs associated with it.

The proposed algorithm achieves this control over the population size by defining a *collective* probability for the population to survive (i.e. to expand or to shrink), together with the *personal* probability of each individual to survive.

The personal survival probability for each $\xi_j \in \xi$ is defined as $p_i = \varphi(\xi_j) / \varphi^*$, where $\varphi(\xi_j)$ is the fitness of ξ_j and φ^* is the maximum fitness within the population. In other words, $\xi_j \in \xi'$ with probability p_i for each $\xi_j \in \xi$.

The size N of the population after selection can decrease (in the extreme case – to one, if just the individual with the maximum fitness survives), remain the same or increase (in the extreme case – to $3.N$, if all parents and

offspring survive). It is desirable that the population size never gets below the initial (i.e. generation-0) population size and the population increase is desirable to be inverse-proportional to the convergence degree of the individuals. The convergence degree can be measured by the ratio between the average fitness and maximum fitness in the population.

The *expected population size* s^E after selection can be roughly approximated by the sum of the survival probabilities of all individuals in the population[1].

The *desired population size* s^D is defined as a linear combination of the two extreme alternatives with coefficients depending on the convergence degree:

$s^D = s^0 c + 3N(1 - c)$,

$c = \varphi' / \varphi^*$,

where s^0 is the desired population size,

N is the current population size,

c is the convergence degree,

φ' is the average fitness and

φ^* is maximum fitness.

The *collective survival probability* p^* is defined as the normalized ratio between the desired and the expected population size ($p^* = s^D / (s^D + s^E)$).

The individual survival probability p_i is then scaled up or down depending on the collective survival probability p^*. In case the population size drops below s^0 in some generation, new random individuals are generated to fill it up to size s^0.

Note that with this algorithm the individual with the maximum fitness survives with probability 1.0, i.e. the proposed selection algorithm always preserves the best individual.

Performance Analysis

The GA proposed in the previous section is convergent, i.e. the fitness of the best individual converges to the global optimum in the solution space as the number of generations tends to infinity. This is a corollary of the properties of the three genetic operators: the selection operator preserves the best individual in the generation with probability 1.0, and the mutation and crossover operators guarantee that every point in the solution space is reachable starting from any randomly selected initial set of points (i.e. individuals in the initial random population). A formal proof of the GA convergence under the above limitations on the properties of the genetic operators can be found in [7]. Convergence is proved by means of homogeneous finite Markov chain analysis.

In order to evaluate the performance of the optimization algorithm, we need to define a performance measure and run a series of statistically significant experiments. The GA proposed in this paper is compared against the most popular classical GA (using Elitist selection) and against the two simplest randomized optimization algorithms, the *Random Search* and the *Random Walk*. In Elitist selection, population size is fixed during the whole optimization process. If the fixed population size is N, Elitist selection simply sorts the individuals in the population in decreasing order of their fitness and preserves the first N of them. The Random Search algorithm generates a finite sequence of random solutions and preserves the solution with the highest fitness found. The Random Walk algorithm starts from a random point in the solution space and on each iteration makes a “move” from the current point to a “neighboring” one (i.e. one that can be reached by applying some mutation operator on the current solution), if this move improves the fitness value.

[1] The exact calculation of the mathematical expectation of the sum of N random variables is significantly more complex and great precision is not needed here.

All experiments were executed using a randomly generated data dictionary and randomly generated queries against it. One and the same simplified cost model and fitness function are used in all experiments. Mutation activity was set to 0.1 and the neighborhood size parameter *k* was set to 6.

A good measure for the performance of the optimization algorithms is the evolution of the maximum fitness as a function of the number of solutions processed. Most performance studies evaluate the evolution of the maximum fitness as a function of the number of generations (for GAs) or iterations (for randomized algorithms) but this metric is inappropriate in our experiments as one of the GAs is characterized by dynamic population size. Experiments with different initial population sizes for the GAs were also executed.

First, we consider the performance of the two GAs and the two randomized algorithms optimizing a query with 7 joined relations. The reason for choosing this particular query size is that it is considered the upper limit for "classical" (or "small") queries and it roughly marks the boundaries of applicability of deterministic query optimization algorithms (for larger queries, deterministic search is not applicable). In fact, this is the largest query whose solution space we were able to fully traverse for a reasonable computation time and so we have the exact values of the global minimum and global maximum of the fitness function.

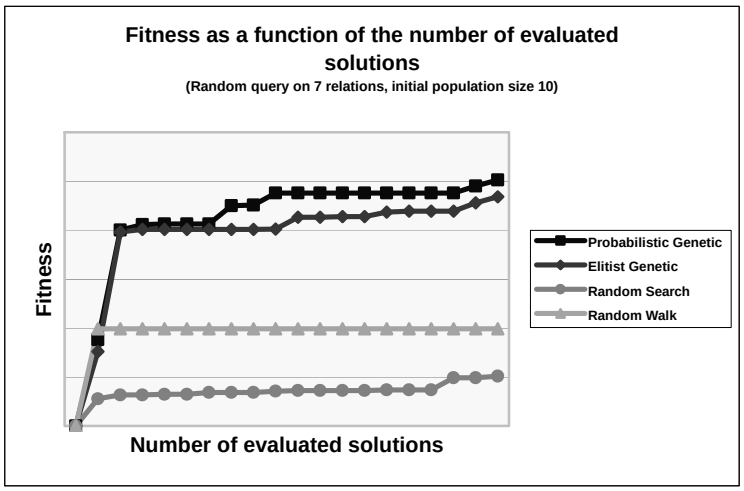

Figure 1. Performance comparison of genetic and randomized optimization algorithms. Random query with 7 joined relations, initial population size 10.

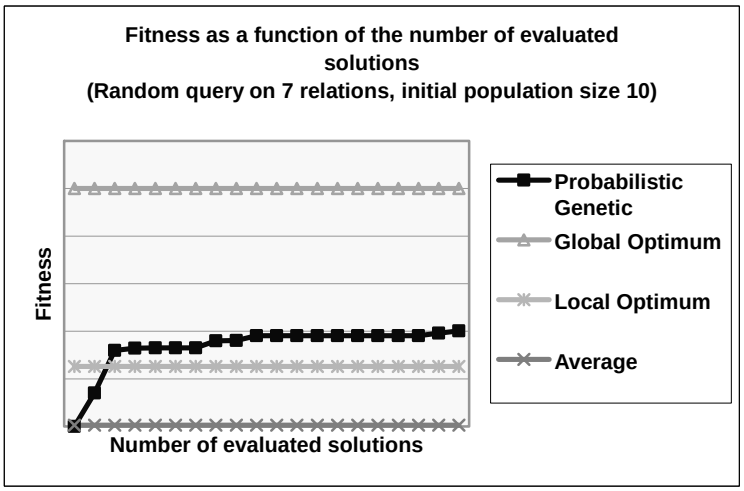

Figure 2. Performance of Probabilistic selection GA. Values compared to the average solution space fitness, a good local optimum and the global optimum.

The two GAs obviously outperform the two randomized algorithms. The random search strategies were easily trapped into local optima and failed to produce solutions that could compete with the best ones found by the GAs. The two GAs show overall similar performance, but the Probabilistic Selection seems a bit better when the number of evaluated solutions gets larger. One pronounced advantage of the Probabilistic Selection is the slower convergence compared to the Elitist selection, which allowed the Probabilistic GA to escape some suboptimal plateaus into which the Elitist GA was trapped. One probable reason is that the populations under Elitist GA are characterized by poorer diversity (very close values of the minimum, average and maximum fitness within the population).

Both GAs performed very well in terms of quality of the solutions found. Figure 2 shows the evolution of the maximum fitness in Probabilistic GA within the boundaries of the average fitness in the solution space and the global optimum. One of the best local optima (into which many of the runs were trapped) is also shown.

Further we test the performance of the four algorithms on a much harder optimization problem – a query with 100 joined relations.

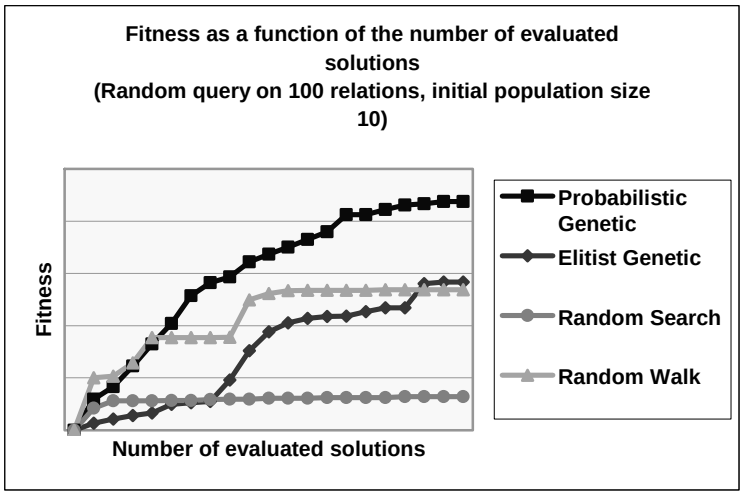

Figure 3. Performance comparison of genetic and randomized optimization algorithms. Random query with 100 joined relations, initial population size 10.

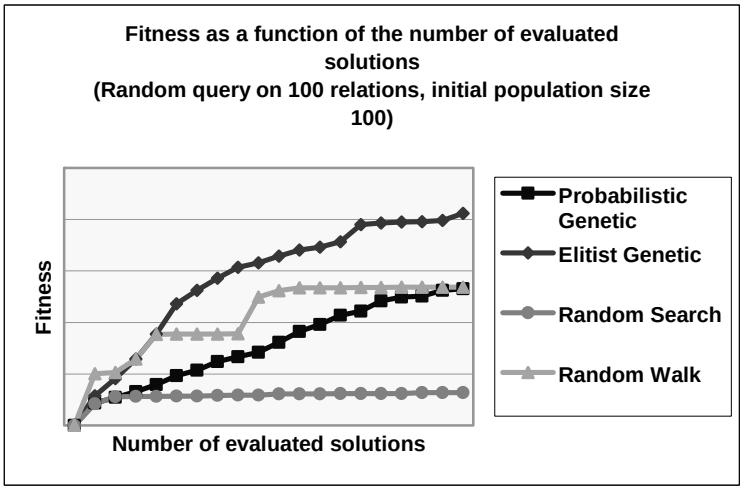

Figure 4. Performance comparison of genetic and randomized optimization algorithms. Random query with 100 joined relations, initial population size 100.

Here the superiority of the two GAs over Random Search is even more pronounced. The Random Walk however performed surprisingly well, with results often comparable to those of the GAs. The relative performance of the

Probabilistic GA and the Elitist GA varied considerably depending on the initial population size, as we can see from the comparison of Figure 3 and Figure 4.

Similar results were observed in experiments with smaller queries also. No rule linking the relative performance of the two GAs and the initial population size could be deduced however – for example, optimizing a query with 10 joined relations, Probabilistic GA was better on initial population size 10, worse on initial population size 20, again better on size 50, etc.

Conclusions and Future Work

In this work an adaptive genetic optimization algorithm is proposed for the join ordering problem. It outperforms the canonical genetic algorithm with classical fixed-size population selection operator such as the Elitist selection in a number of query optimization experiments. Considering the facts that

- Probabilistic Selection is expected to be faster than the Elitist – computational complexity $O(N)$ versus $O(N.\log(N))$
- the performance of the two selection operators are comparable and in many cases Probabilistic selection is better than Elitist
- Probabilistic Selection maintains a population with better diversity and more easily escapes suboptimal plateaus in the solution space,

the proposed optimization algorithm is a viable contender.

Overall, the results unconditionally prove the applicability of genetic optimization algorithms to the join ordering problem. GAs prove to be a competitive alternative to deterministic optimization algorithms even for small solution spaces.

One direction for future work would include experimenting with adaptive mutation and crossover operators. It is reasonable to expect further performance improvements, as it is suggested by a number of recent researches. The proposed algorithm can also be used as a basis for a hybrid optimization algorithm incorporating certain domain-specific heuristics and randomized local search techniques.

Bibliography

[1] A. Eiben, E. Marchiori, V. Valkó. Evolutionary Algorithms with on-the-fly Population Size Adjustment. In: Proc. of the 8th International Conference on Parallel Problem Solving From Nature, 2004.

[2] T. Haynes. A comparison of random search versus genetic programming as engines for collective adaptation. In: Proc. of the ACM Symposium on Applied Computing, 1997.

[3] M. Stillger, M. Spiliopoulou. Genetic programming in database query optimization. In: Proc. of the 1st Annual Conference on Genetic Programming, 1996.

[4] M. Steinbrunn, G. Moerkotte, A. Kemper. Optimizing join orders. In: Report MIP 9307, Universität Passau, 1993.

[5] K. Bennett, M. Ferris. Y. Ioannidis. A genetic algorithm for database query optimization. In: Proc. of the 4th International Conference on Genetic Algorithms, 1991.

[6] Y. Ioannidis, Y. Kang. Randomized algorithms for optimizing large join queries. In: Proc. of the ACM, 1990.

[7] G. Rudolph. Convergence Analysis of Canonical Genetic Algorithms. In: IEEE Transactions on Neural Networks, 1994.

Author's Information

Stoyan Vellev – PhD student, Faculty of Mathematics and Informatics, Sofia University, 7 Raiko Alexiev Str, bl. 30, Sofia-1113, Bulgaria; e-mail: stoyan.vellev@sap.com