МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2005
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2005
Proceedings of the Thirty Fourth Spring Conference of
the Union of Bulgarian Mathematicians
Borovets, April 6–9, 2005

ADAPTIVE GENETIC OPTIMIZATION ALGORITHM FOR THE JOIN ORDERING PROBLEM

Stoyan M. Vellev

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, which outperforms in a series of experiments several classic genetic algorithms and proves to be a viable alternative to existing non-deterministic optimization approaches.

1. 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. This is done by a dedicated RDBMS module, called 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 modern 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 for 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*.

2. 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 $R^q = \{R_i | R_i$ is referenced in $Q\}$, $R^q \subseteq \mathcal{R}$ and $P^q = \{p_i(R^i_j, R^i_k) | p_i$ 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 join ordering problem is NP-complete.

3. Related work. Due to the inapplicability of classic deterministic optimization algorithms (different variations of the classic dynamic optimization with pruning) to the join ordering problem for large queries (where *large* is usually defined as queries with 10 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 (II) and Simulated Annealing (SA), as well as a combination of the two [6]. Though the effectiveness of randomized algorithms strongly depends on shape of the solution space, they generally prove to be a possible alternative to deterministic algorithms for large queries.

The comparison between randomized and genetic algorithms is definitely in favor of the latter for large-size problems [2].

Genetic algorithms 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 correct solution encoding. Despite these shortcomings, the achieved results are promising. Later in [3] some of these disadvantages have been overcome.

Currently there is only one non-experimental genetic SQL query optimizer – 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.

4. The algorithm. We introduce an adaptive genetic algorithm as an efficient solution to the optimal join ordering problem.

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

$$\Theta(\xi_i) = \Theta(R_{i1} \bowtie_{p1} R_{i2} \bowtie_{p2} \cdots \bowtie_{pn-2} R_{in-1} \bowtie_{pn-1} R_{in}) \rightarrow ((i_1, p_1), (i_2, p_2), \ldots, (i_n, p_n))$$

Mutation. The mutation operator M transforms an individual ξ_i 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:

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

The *mutation rate* μ is the probability of an individual ξ_i to mutate on each generation, i.e. $\mathrm{M}(\xi_i) \equiv \xi_i$ with probability $(1-\mu)$. 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_i^g and ξ_j^g to obtain two new generation-$(g+1)$ individuals ξ_i^{g+1} and ξ_j^{g+1}. 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 in relation to the locus) and the rest of the chromosome is filled up with the missing genes in the order they are encountered in the second parent. This guarantees both structural and functional similarity of the children with both their parents.

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

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

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

$$\Sigma(\xi) = \Sigma(\{\xi_1,\xi_2,\ldots,\xi_k\}) = \{\xi_{i1},\xi_{i2},\ldots,\xi_{im}\}, \quad m \leq k.$$

We propose an algorithm called *probabilistic selection with adaptive population size*. If $\xi_i \in \xi$, then $\xi_i \in \xi$' with probability $p_i = \phi(\xi_i)/\phi^*$, where $\phi(\xi_i)$ is the fitness of ξ_i and ϕ^* is the maximum fitness within the population. Then the expected size of the population after selection can be roughly approximated by the sum of survival probabilities of each individual in the population. The ratio c between the current average and the current maximum fitness is used as a measure for the convergence degree of the population. The desired population size is defined as $s^0c+3N(1-c)$, where s^0 is the initial population size, N is the current size and c is the convergence degree. The individual survival probability p_i is then scaled up or down depending on the ratio between the desired and the expected population size. In case the population size drops below s^0 on some generation, new random individuals are generated to fill it up to size s^0.

5. 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 classic fixed-size population selection operators such as the *Elitist* and the *Roulette* selection, both by speed and by quality of the generated solutions. The results of these experiments will be presented in a further paper.

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.

REFERENCES

[1] A. Eiben, E. Marchiori, V. Valkó. Evolutionary Algorithms with on-the-fly Population Size Adjustment. 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. Proc. of the ACM Symposium on Applied Computing, 1997.

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

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

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

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

Stoyan Miltchev Vellev
7, Raiko Alexiev Str, bl. 30
1113 Sofia, Bulgaria
e-mail: stoyan.vellev@sap.com

АДАПТИВЕН ГЕНЕТИЧЕН АЛГОРИТЪМ ЗА ЗАДАЧАТА ЗА ПОДРЕЖДАНЕ НА СЪЕДИНЕНИЯ

Стоян Милчев Велев

Намирането на оптималния ред на съединенията при изпълнение на заявка към релационна база от данни представлява комбинаторна оптимизационна задача, за която детерминистичното търсене с пълно изчерпване е неприемливо за заявки с голям брой съединени релации. В настоящата работа се предлага адаптивен генетичен алгоритъм с динамичен размер на популацията, който превъзхожда в редица експерименти няколко класически генетични алгоритми и доказва, че е жизнеспособна алтернатива на съществуващите недетерминистични подходи.