XI Jornadas de Ingeniería del Software y Bases de Datos
JISBD 2006
José Riquelme - Pere Botella (Eds)

INTENSIVE CROSSOVERS: IMPROVING QUALITY IN A GENETIC QUERY OPTIMIZER

Victor Muntés Mulero[1], Josep Aguilar Saborit[1], Calisto Zuzarte[2] and Josep Lluís Larriba-Pey[1]

1: DAMA-UPC[1]
Universitat Politècnica de Catalunya (UPC)
C/Jordi Girona 1-3 08034 Barcelona
e-mail: {vmuntes, jaguilar, larri}@ac.upc.edu, web: http://www.dama.upc.edu

2: IBM Canada Ltd., IBM Toronto Lab
8200 Warden Ave, Markham, Ontario, Canada. L6G1C7
e-mail: calisto@ca.ibm.com

Key words: Large Join Query Optimization, Genetic Programming, Search Strategies

Abstract. *Database schemas and user queries are continuously growing with the need for storing and accessing large amounts of structured information. Among the several proposals to deal with the Large Join Query Problem, genetic optimizers have been shown to be a competitive approach.*

We propose a new search strategy to improve the quality of genetic query optimizers. We call our technique Intensive Crossovers (IC) and it shows that, in terms of quality of the results, it is worthier to spend more time creating extra child plans locally in a crossover operation than to focus on crossover operations on a lot of different execution plans. After the first analysis of IC, we propose an improved technique called Increasing Intensive Crossovers (IIC). The idea behind this improvement is to speed-up the convergence of IC.

All in all, we show that the search strategy of choice is paramount to determine the quality and convergence of a genetic query optimizer, opening a new line of research oriented to unlink genetic optimizers from their dependency on the random effects of, both, the initial population and the random decisions taken through the optimization process.

1 Introduction

Advanced applications like SAP often require to combine a large set of tables to reconstruct complex business objects. For instance, the SAP schema may contain more than 20000 relations and may join more than 20 of these in a single SQL query. It is the task

[1]Research supported by the IBM Toronto Lab Center for Advanced Studies and UPC Barcelona. The authors from UPC want to thank Generalitat de Catalunya for its support through grant GRE-00352.

of the query optimizer to transform a SQL statement into a query execution plan (QEP), determining the cost of each alternative QEP and selecting the cheapest one.

State-of-the-art query optimizers, which typically employ dynamic programming [8] usually fall back to greedy algorithms when the plan space requirement to store various intermediate subplans is too large. Moreover, greedy algorithms or any other methods using heuristic based planning do not consider the entire search space and may overlook the optimal plan.

Randomized search techniques, like genetic programming, remedy the exponential explosion of dynamic programming techniques by iteratively exploring the search space and converging to a nearly optimal solution. The Carquinyoli Genetic Optimizer (CGO) has been presented recently as the first sound and complete genetic optimizer [5]. To the best of our knowledge, CGO is the only genetic optimizer studied and tested against a commercial optimizer, proving its competitiveness regarding speed and quality.

In this paper we propose a new search strategy that provides the means to improve the quality of the plans obtained by genetic query optimizers. Our technique is called Intensive Crossovers (IC) and it shows that spending more time locally creating extra child plans in a crossover operation is wiser than, in opposition, focusing on what we call *extensive crossovers*, i.e. combining a lot of different execution plans in the population generating a reduced number of children per crossover (typically 2 children). The results show that we obtain, on average, after a certain amount of time, execution plans which have a cost between 3 and 20 times lower than the best cost obtained by the original execution of CGO.

However, IC does not converge fast, needing a warm-up period to outperform the original CGO. In order to mitigate this effect, i.e. to improve the convergence of IC, we propose a further improvement by dynamically increasing the number of internal crossovers during the execution. We call this improvement Increasing Intensive Crossovers (IIC).

The rest of this paper is organized as follows. In section 2, we introduce the reader to the basic concepts of Genetic Programming. Section 3 presents a detailed description of our new search strategy. Section 4 shows the improvements achieved by the application of our technique and propose an enhancement in order to achieve better speed of convergence. Section 5 references some related work. Finally, Section 6 presents the conclusions extracted from our work.

2 Genetic Programming in Query Optimization

Genetic programming (GP) has been exhaustively explored since the publication of Koza's book in 1992 [3]. The basic idea is to obtain a best fit solution, called program originally, to solve a problem using evolutionary methods.

The basic behavior of this type of algorithms is as follows. An initial set of programs is created from scratch. In this paper we represent programmes as tree structures, since they are the most suitable approach given that QEPs in DBMSs are usually tree-shaped. This set is also called the *initial population*.

Once the initial population is created, we iteratively apply a set of genetic transformations on the members in the population. The primary transformation operators are *crossover* and *mutation*. The former works by changing two (or more) programs (or tree structures) combining them in some manner; the latter by modifying a single tree structure. Each iteration of the algorithm is called a *generation*. At the end of each generation, a third genetic operation called *selection* is applied in order to eliminate the worst fitted members in the population.

After applying these operations the algorithm obtains the next generation of members. A *stop condition* ensures that the algorithm terminates. Once the stop criteria is met, we take the best solution from the final population.

One of the typical applications for this type of algorithm is to solve optimal path search problems. In these problems, each member in the population represents a path to achieve a specific objective and has an associated cost.

Query optimization can be reduced to a search problem where the DBMS needs to find the optimum QEP in a vast search space. Each execution plan can be considered as a possible solution program for the problem of finding a good access path to retrieve the required data. Therefore, in a *genetic optimizer*, every member in the population is a valid execution plan. Intuitively, as the population evolves, the average plan cost of the members decreases.

3 Intensive Crossovers

A typical inherent problem in GP is the fact that, after some generations, the genetic operations become very disruptive, tending to create new members with self lethal traits, which are typically discarded in the next selection operation. In order to show the disruptive trends of crossover operations, we calculate the percentage of improvement using equations (1) and (2) below where p_{p1} and p_{p2} are the parent plans, p_c is one of the children generated by those plans in the crossover operation, and $cost(x)$ is a function returning the cost associated to x, where x is a QEP. The idea behind the equations is to calculate whether the new plan is better than the average cost between both parent plans.

$$r_{imp} = \frac{2 \cdot cost(p_c)}{cost(p_{p1}) + cost(p_{p2})} \tag{1}$$

$$\%_C = \begin{cases} (1 - r_{imp}) \cdot 100 & \text{if } r_{imp} \leq 1 \\ (\frac{1}{r_{imp}} - 1) \cdot 100 & \text{if } r_{imp} > 1 \end{cases} \tag{2}$$

Figure 1 shows the average maximum and minimum efficiency for a crossover operation, through several generations, executed using 15 random queries involving 50 relations. The plot clearly shows that the efficiency of crossover operations rapidly decays from generation 25 up to generation 80. From generation 80 onwards, it can be observed that gains are merely marginal. On the other hand, the disruptive effects of the operation are noticeable, i.e. the probability of obtaining a bad QEP is higher than the probability of

obtaining a good plan. After generation 110, the algorithm has lost all the diversity in the population, and crossovers are just crossing plans which are the same.

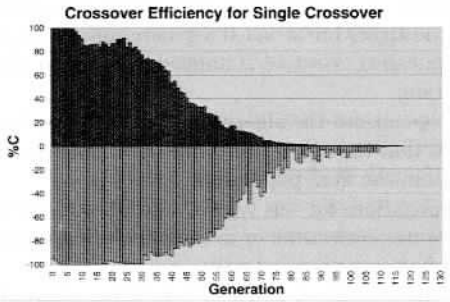

Figure 1: Average maximum and minimum crossover efficiency for 50 relations.

In order to increase the quality of genetic operations, we propose a new search strategy for genetic query optimizers called Intensive Crossovers (IC). IC has the objective to reduce the disruptive effects of crossover operations by intensifying the local search. This way, as depicted in Figure 2, instead of applying single crossover operations as in the original CGO, our strategy proceeds as follows:

1. Two individuals are randomly chosen from the population
2. Then, N random crossover operations are performed on the same parents, producing $2N$ offsprings. We call these operations *internal crossovers*.
3. The associated cost, or fitness function, is calculated for every generated child and only the best two QEPs are kept. The remaining are directly discarded.

Naturally, increasing the amount of time spent for a single crossover operation, causes an increase in the amount of time used for a generation. However, the leading idea behind the algorithm is the fact that the reduction of the destructive effects of the crossover operations compensate for the increase of execution time.

4 Analysis and Improvements

In the following subsections we study the behavior of our technique, comparing it to the original execution of CGO. The methods used to create random databases and queries for the analysis are detailed in [6].

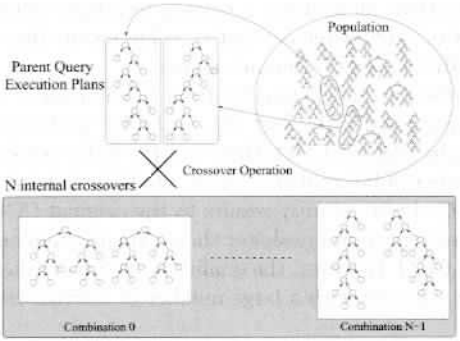

Figure 2: Intensive Crossover.

4.1 Intensive Crossovers

The basic idea of this section is to show that investing time using Intensive Crossovers, thus elongating the time used for a genetic operation, has a positive impact in the quality of the obtained QEP.

We have tested IC using random large join queries which involved 30, 50 and 100 relations each. Due to a lack of space we only present the results for 50 joins unless stated, however, our results show that conclusions can be extended to all the cases.

We have run CGO several times to experimentally decide the best values for the following parameters. We run the same test on 15 randomly created queries executed on 15 random databases. For each query we execute the original CGO and all the techniques 5 times and calculate the average best cost. We execute 50 crossover operations per generation. To better analyze the effects of IC, mutations have been disabled in CGO. We have fixed the execution time to 8 minutes.

For each query, we compare the original execution of CGO with the execution for the same initial population using respectively 2, 4, 8, 16 and 32 internal crossovers.

In order to fairly compare results, we use the following formula to compute the scaled cost:

$$ScaledCost = \begin{cases} C_{orig}/C_{newTech} - 1 & \text{if } C_{orig} \geq C_{newTech} \\ 1 - C_{newTech}/C_{orig} & \text{if } C_{orig} < C_{newTech} \end{cases} \quad (3)$$

where C_{orig} represents the best cost obtained by the original implementation of CGO and $C_{newTech}$ represents the best cost achieved by the technique to be tested. In this way, the scaled cost in formula (3) allows to obtain an average from the execution of different queries and databases and it is centered in 0.

Figure 3 shows the scaled best cost for different numbers of internal crossovers. We consider the execution of the original CGO to be the baseline. Consequently, a positive

value in the plot means that, on average, a certain technique improves over the original after a specific amount of time, while a negative value means that the technique cannot achieve a better cost after a specific amount of time.

The plot clearly shows that, independently from the number of internal crossovers, IC eventually improves the quality of the best obtained QEP with scaled costs ranging from 2.5 to 18 compared to the original CGO. However, the differences in the scaled costs are conditioned by the number of internal crossovers. On the one hand, with a small number of internal crossovers IC obtains similar results to the original CGO after 30 s. A larger number of internal crossovers converge slower than a smaller number and need more time to improve the original CGO. However, the quality of the execution plans obtained during the execution is clearly superior with a large number of internal crossovers.

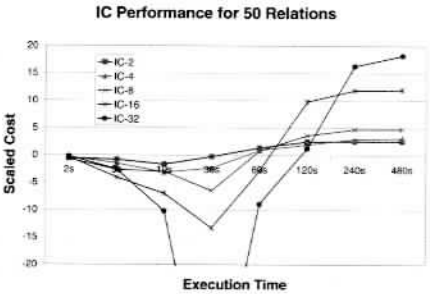

Figure 3: Scaled best cost evolution for different internal crossover numbers.

Figure 4 shows the actual number of generations executed on average, in 8 minutes, for each specific number of internal crossovers for queries involving 30, 50 and 100 relations. The objective of this plot is to show that, since IC increases the number of crossover operations per generation, the time needed to execute a generation is proportionally increased, thus, the number of generations executed in a period of time is reduced. Specifically, we can observe that the number of generations executed using 2 internal crossovers per crossover operation is roughly half of the generations executed by the original. In general, we observe that using N internal crossovers the number of generations executed is the number of generations executed by the original divided by N. This effect is due to the fact that, for each crossover operation, we are repeating the same operation N times, thus multiplying the time by N. These results indicate that, although the number of generations and the number of genetic operations have decreased significantly, improving the efficiency of crossover operations, by reinforcing local search, helps the algorithm to improve the quality of results.

This observation implies that the use of IC reduces the disruptive effects of the crossover operations. Figure 5 corroborates this idea by showing the average maximum and mini-

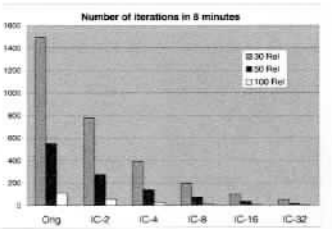

Figure 4: Number of generations executed in eight minutes for different number of internal crossovers ranging from 2 to 32.

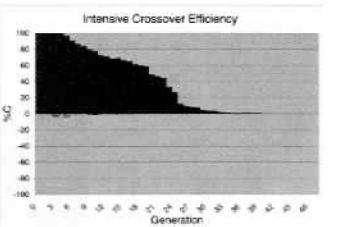

Figure 5: Average maximum and minimum crossover efficiency for 50 relations using IC (N = 16).

mum gain obtained in a crossover operation using equations (1) and (2) and 16 internal crossovers per intensive crossover for queries involving 50 relations. Results show that, in general, crossover operations in the context of IC do not introduce lethal properties into the new offsprings.

4.1.1 Convergence of IC

IC is unable to converge as fast as CGO during the first generations. In addition, the slow convergence is even more noticeable as the number of internal crossovers increases. The reason is explained in Figure 6 where we can see the average number of QEPs which, although having a better cost than the average between the costs of their parents, are discarded by the internal selection carried out during an intensive crossover. From the results we see that IC spends a lot of time during the first generations creating good execution QEPs which are discarded without having the chance to survive. For this reason, as previously observed in Figure 3, IC does not achieve better results than CGO during the first generations. The basic problem is that CGO also obtains good quality when it employs crossover operations during the first generations, but in less time than IC. However, as observed in Figure 1, after several generations the disruptive effects in crossover operations for CGO are very notorious while, using IC, they are almost eliminated. Note that Figures 1 and 5 cannot be directly compared since each generation using a number of internal crossovers equal to 16 is equivalent to 16 generations in the original case.

4.2 Increasing Intensive Crossovers

We propose a further improvement on IC in order to soften the slow convergence of the algorithm during the first generations of IC when using a large number of internal

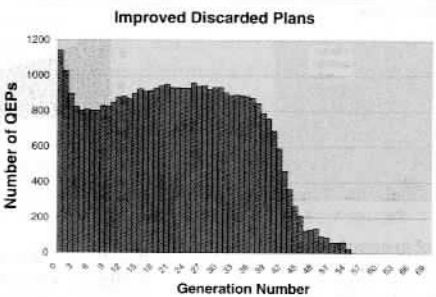

Figure 6: Average number of plans discarded although their cost improves the average of their parent QEPs using IC-16.

crossovers. The previous subsection shows that the effects of IC can only be noticed after some generations because IC is wasting time during the first generations, discarding good QEPs. In this section we propose an enhanced version of IC called Increasing Intensive Crossovers (IIC). As opposed to IC, IIC dynamically increases the number of internal crossovers periodically after a fixed number of iterations.

For this experiment, IIC starts executing 2 crossovers per intensive crossover, which is increased exponentially every 5 generations. We have fixed the maximum number of internal crossovers to 32 to avoid degradation due to excessively long intensive operations.

In Figure 7, we show the comparison between IIC and the previous results obtained for IC-2 and IC-16. We can make two major observations: first, IIC does not invest a lot of time creating and discarding good QEPs during the first generations, thus it obtains the same speed of convergence as IC-2. Second, the quality of the results clearly improves the quality of those cases with a small number of internal crossovers, like IC-2. Of course, the results obtained by the increasing version cannot reach the quality of IC-16 since the quality of the QEPs obtained during the first generations is worse for IIC.

5 Related Work

Several attempts have been carried out in order to diminish the Large Join Query Problem. Randomized search techniques remedy the exponential explosion of dynamic programming techniques by iteratively exploring the search space and converging to a nearly optimal solution. Genetic algorithms are a randomized search technique [2] modelling natural evolution over generations using crossover, mutation and selection operations. Optimizing queries involving a large number of joins using genetic algorithms was introduced by Bennet el al. [1] and tested later by Steinbrunn et al. [9] showing that it is a very competitive approach. Genetic programming applied to query optimization was first introduced in [10]. Stillger, based on these previous works, presents a genetic

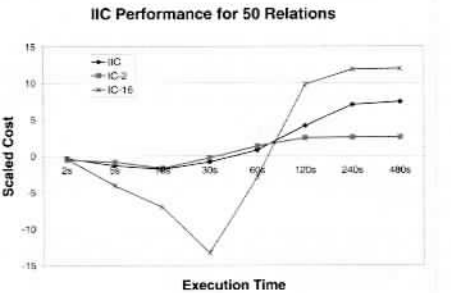

Figure 7: Comparison between IIC, IC-2 and IC-16.

programming-based optimizer that directly uses execution plans as the members in the population, instead of using chromosomes. A first genetic optimizer prototype was created for PostgreSQL [7], but its search domain is reduced to left-deep trees and mutation operations are deprecated, thus bounding the search to only those properties appearing in the execution plans of the initial population. Later in [5], the Carquinyoli Genetic Optimizer (CGO), which is also based on GP is presented and tested, for the first time, against a well-known commercial optimizer. Muntés et al. also present in [4] a study to establish criteria in order to parameterize a genetic optimizer.

Tackett proposes a method called *brood recombination* in his thesis [11] in order to reduce the disruptive effects of crossover operations. Brood recombination is based on the observation of various animal species in nature which produce more offspring than those expected to live.

6 Conclusions

In this paper, we show that Intensive Crossovers have an impact on the quality of genetic query optimizers, by assuring intensive genetic operations. Our analysis shows that, although intensive operations are unnecessary at the beginning of the optimization process, they become increasingly more necessary along the time. Therefore, we show that our new technique called Increasing IC improves quality compared to the original CGO avoiding the problems of IC regarding the speed of convergence during the first generations of plans.

Moreover, our achievements make a step further in the search of general techniques that unlink genetic optimizers from their dependency on the random effects of, both, the initial population and the random decisions taken through the optimization process.

All in all, our work shows that studying search strategies is still an open and fruitful research line which may lead genetic query optimizers towards the solution of the Large Join Query Problem.

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] J. Holland. Adaption in natural and artificial systems. The University of Michigan Press, Ann Arbor, 1975.

[3] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT press, Cambridge, MA, 1992.

[4] V. Muntés-Mulero, J. Aguilar-Saborit, M. Prez-Casany, C. Zuzarte, and J.-L. Larriba-Pey. Parameterizing a genetic optimizer. In *Proceedings of the International Conference on Database and Expert System Applications (To be published)*, September 2006.

[5] V. Muntés-Mulero, J. Aguilar-Saborit, C. Zuzarte, and J.-L. Larriba-Pey. Cgo: a sound genetic optimizer for cyclic query graphs. In *Proceedings of the International Conference on Computer Science*, pages 156–163, May 2006.

[6] V. Muntés-Mulero, J. Aguilar-Saborit, C. Zuzarte, V. Markl, and J.-L. Larriba-Pey. Genetic evolution in query optimization: a complete analysis of a genetic optimizer. Technical Report UPC-DAC-RR-2005-21, Departament d'Arquitectura de Computadors. Universitat Politecnica de Catalunya (http://www.dama.upc.edu), 2005.

[7] PostgreSQL. http://www.postgresql.org/.

[8] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on the Management of Data*, pages 23–34. ACM Press, 1979.

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

[10] Michael Stillger and Myra Spiliopoulou. Genetic programming in database query optimization. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 388–393, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[11] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, Department of Electrical Engineering Systems, USA, 1994.