# Optimization of Large Join Queries: Using Heuristics to generate Initial Population of Genetic Programming

***Ko Ko Naing (kokonaing@gmail.com)***
***Tang Van To (tvto@scitech.au.edu)***
*Faculty of Science and Technology*
*Assumption University, Bangkok 10240, Thailand.*

**ABSTRACT** – Optimizing queries is one of the hardest problems in database field since exhaustive searching of an optimal query execution plan in a large search space is impractical. Genetic Programming is an evolutionary algorithm that makes use of stochastic decision making method for optimization problems. In pure Genetic Programming approach, initial population is generated randomly without any considerations on the generated individuals. We propose the use of heuristics in generating initial population for the genetic runs so that we are starting the process with relatively good individuals, expecting to obtain an optimal query execution plan earlier.

**KEYWORDS –** Query Optimization, Genetic Programming, Large Join Queries, Heuristics

## 1 INTRODUCTION

Query optimization has been an active research topic for decades. Several researchers had proposed various techniques to optimize queries. Among them, the famous ones are query optimization using Simulated Annealing (SA) [IOANNIDIS and WONG 87], using Iterative Improvement (II) [SWAMI and GUPTA 88], using Two Phase Optimization (2PO - a combination of II and SA) [IOANNIDIS and KANG 90]. In addition to the above mentioned randomized algorithms, many researchers applied heuristics while implementing the above algorithms to efficiently generate good starting points for the future run of the algorithms. KBZ heuristic approach proposed in [KRISHNAMURTHY et al 86] and Augmentation and Local Improvement heuristics proposed in [SWAMI 89] are popular in optimizing queries using heuristics. Predefined heuristics significantly made randomized algorithms run more efficiently in those researches.

When Genetic approaches such as Genetic Algorithm and its descendent Genetic Programming become popular in 1990's, many researchers tried to apply Genetic approaches into the query optimization area. Genetic Algorithm is applied in [BENNETT et al 91] and Genetic Programming is applied in [STILLGER et al 96]. We try to apply heuristics that were implemented for randomized algorithms as mentioned above and compare them with the ordinary Genetic Programming approach proposed in [STILLGER et al 96]. The purpose of applying heuristics is to provide the optimizer with the initial population containing popular individuals and to expect to find optimal solution more efficiently than the ordinary Genetic Programming approach.

## 2 GOALS AND OBJECTIVES

The main objective of this research is to apply the Genetic Programming in solving query optimization problem with the help of various heuristics that had been proposed.

Other objectives are to explore the possible problem areas in which Genetic Programming can be applied and to contribute in the area of query optimization using one of the Genetic approaches and heuristics used for join relational operator. Not only the research needs to apply the Genetic Programming to solve the query optimization problem, but the fitness function based on the various cost estimation methods for join algorithms also needs to be formularized to evaluate each join node of each chromosome in the population.

## 3 GENETIC PROGRAMMING

Genetic Programming is the automated learning of computer programs. GP's learning algorithm is inspired by the theory of evolution and our contemporary understanding of biology and natural evolution [BANZHAF et al 97].

The following is a typical algorithm of Genetic Programming:

```
[Start] Define terminal set, function set, fitness
function and GP parameters such as population size,
maximum individual size, etc. Encode chromosomes into
tree structure.

[New population] Create a new population by repeating
the following steps until the new population is
completely created.

    [Selection] Evaluate the fitness values of the
    chromosomes. Select two parent nodes from a
    population according to their fitness values (the
    better the fitness value, the better chance to be
    selected).

    [Crossover] With a crossover probability,
    crossover the parents to form new offspring
    (children). If no crossover was performed,
    offsprings are the exact copy of parents.

    [Mutation] With a mutation probability, mutate
    new offspring at each node.

[Accepting] Place new offsprings in the new
population.

[Replace] Use new generated population for further
run of the algorithm.

[Test] If the end condition is satisfied, stop, and
return the best solution in current population.

[Loop] Go to step 2.
```

## 4 PROPOSED SOLUTION

Since a query execution plan is typically expressed using query tree, which is indeed a tree (specifically binary tree) structure, chromosomes in this research use binary tree data structure. Query execution plan (QEP) satisfies the structural requirements of the Genetic Programming methodology on a tree representation of an abstract computer program as mentioned in [STILLGER et al 96].

### Crossover Operator

Two subtrees are selected from the two trees that are selected to be applied the crossover operator of this GP model. Then the two subtrees are exchanged between the two parent trees. Figure 4-1 represents the query graph from which the chromosomes in the example derived.

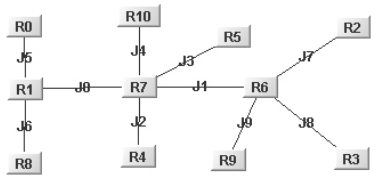


**Figure 4-1 A Query Graph.**

Suppose the two chromosomes representing the following two QEPs are to be crossovered according to the specified subtrees S1 and S2 as in figure 4-2.

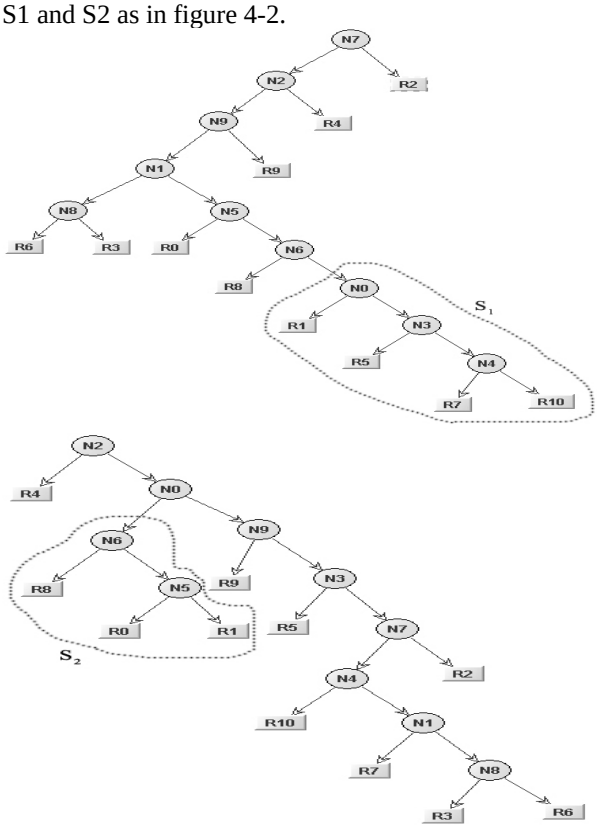


**Figure 4-2 Two selected subtrees for crossover operator.**

After the crossover operator is applied to the two chromosomes provided in figure 4-2, the resulting offspring are generated as in figure 4-3. The original structure of the two parent chromosomes still remain in the offspring.

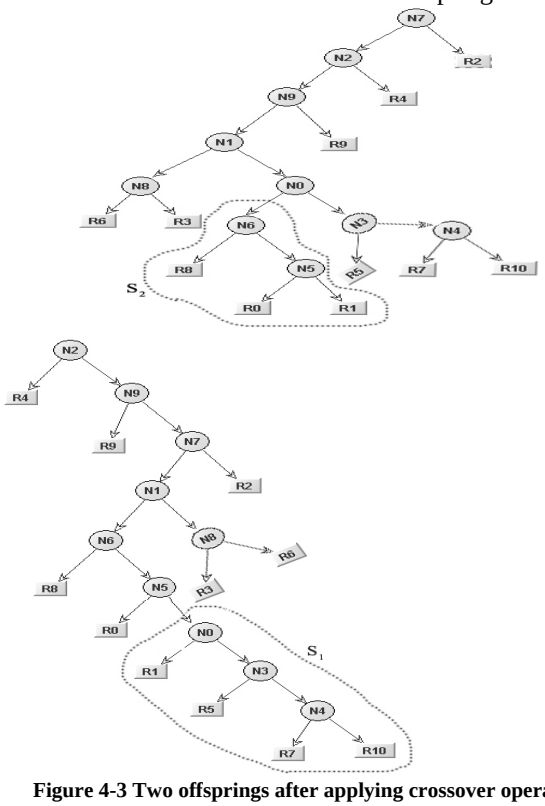


**Figure 4-3 Two offsprings after applying crossover operator.**

We can clearly see that the generated offsprings are popular, because both of them contain the exact same relations and join nodes as in the query graph described in figure 4-1.

**Mutation Operator**

The mutation operator applied in this research works as follows. When a node containing the two leaves (relations) is selected, another node is selected until a node containing at least one non-leaf node is found. The join algorithm is also changed according to the execution order of the rearrangement result. Figure 4-4 shows the chromosome that is to be mutated at the randomly selected node, which is N7.

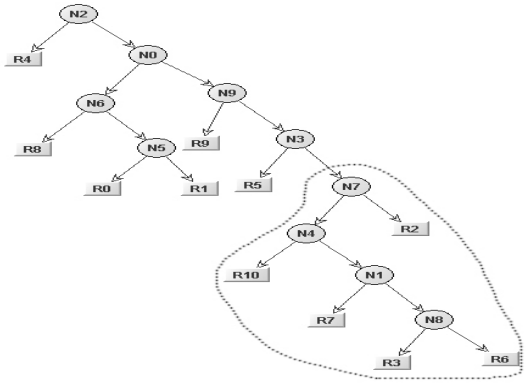


**Figure 4-4: Randomly selected node for mutation operator.**

The nodes under N7 are splitted from the main tree, and nodes N8, N1, N4 and N7 are randomly shuffled to build a new subtree. After the mutation operator is applied to the chromosome, the resulting offspring are generated as in figure 4-5.

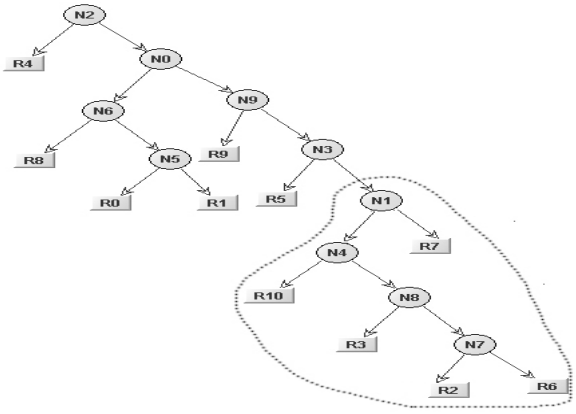


**Figure 4-5: The offspring after applying mutation operator.**

We can clearly see that the offspring generated by applying the mutation operator is popular because the offspring contains the exact same relations and join nodes as in the query graph described in figure 4-1.

**Applying Heuristics to the Initial Population**

The heuristics proposed in [SWAMI 89] are applied when generating the initial population for the Genetic runs, making the Genetic runs start with very good chromosomes compared to randomly generated chromosomes. The heuristics proposed in [SWAMI 89] is called augmentation heuristics. The way it works is as follow. The first relation is randomly picked up from the query graph. And the edges of the relation, which are the join edges, are evaluated based on the join selectivity or the size of the intermediate result. The edge with the smallest value is selected as the first join node to process. And then the relation connected is added to the set of relations from which the query graph can walk through.

Technically, let S be the set of relations that have already involved in selection process and T be the set of relations from which the next relation in the ordering is to be selected. At the beginning of the selection process, S contains only one relation

which is randomly picked by from the query graph. Then the augmentation heuristic generates the rest of the join order as below.

```
firstRelation := randomly picked up from the query graph;
node_list := {}; // the list of the node that is to be processed
S := { firstRelation };
T := {all relations in the query graph except firstRelation };
for(i := 1; i < number_of_relations; i := i + 1) {
        nextRelation := chooseNext(S);
        S := S + { nextRelation };
        T := T - { nextRelation };
        node := getJoinNode(S, nextRelation);
        Add node to the node_list
}
```

The `chooseNext(S)` method chooses the next relation which is accessible from the relations in `S`, according to the criterion defined. The `getJoinNode(S, nextRelation)` method is to obtain a join node that contains `nextRelation` and a relation in `S`. The `node_list` contains the join nodes in ascending order to build a chromosome representing a query tree. In this research, two criteria are used to order the join nodes in the `node_list`: join selectivity and size of the intermediate result. When join selectivity is used, the join node with the first relation that has the lowest value of join selectivity is chosen to be processed first and then the second lowest one and so on. When the size of the intermediate result is used, the join node with the first relation that has the lowest value of the size of the intermediate result is chosen to be processed first and then the second lowest one and so on.

## 5 Results and Discussions

The experimental design used in this research is adopted from the design used in [IOANNIDIS and KANG 90]. We made the same assumptions about uniform distribution of values and independence of values in join attributes. However, three database catalogs setup in this research vary a bit from those in [IOANNIDIS and KANG 90].

### Benchmark Testing

*An Example with 10 joins*

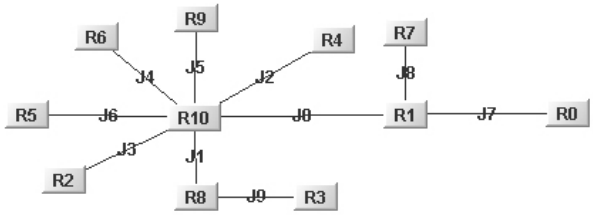


**Figure 5-1: Sample Query Graph**

The following are the best chromosomes from 500th generation of the genetic run using random population, initial population based on intermediate result size and join selectivity respectively. Clearly, we can see that all of them correspond to the query graph shown in Figure 5-1.

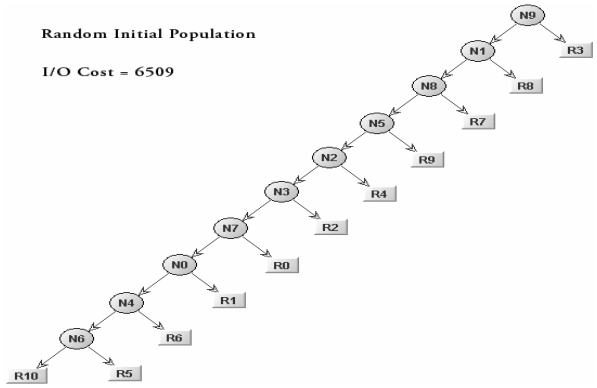


**Figure 5-2: The Best Chromosome from 500th Generation using Random Initial Population**

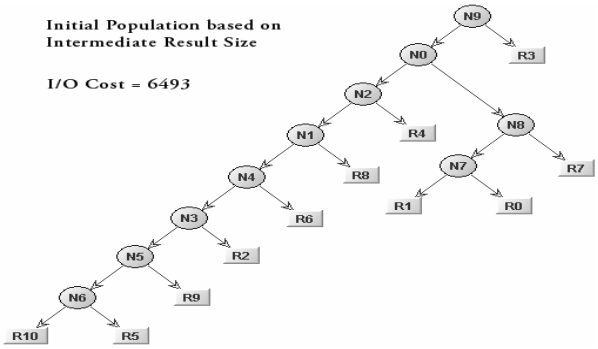


**Figure 5-3: The Best Chromosome from 500th Generation using Initial Population based on Intermediate Result Size**

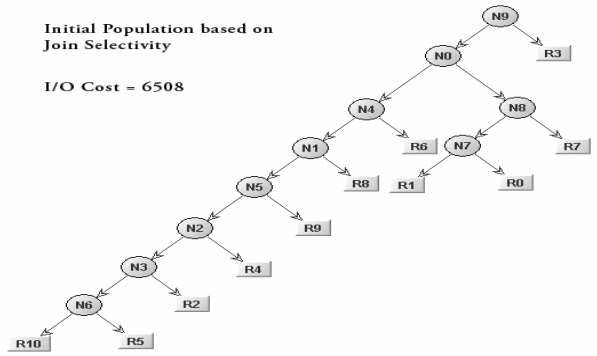


**Figure 5-4: The Best Chromosome from 500th Generation using Initial Population based on Join Selectivity**

*Experiments with 10 joins*

Amount of joins = 10

Database catalog = "Catalog1" for Experiment 1 to 3,
"Catalog2" for Experiment 4 to 6,
"Catalog3" for Experiment 7 to 9

Number of queries = 10 (taken best of them)

Population size = 100

Maximum number of generations = 500

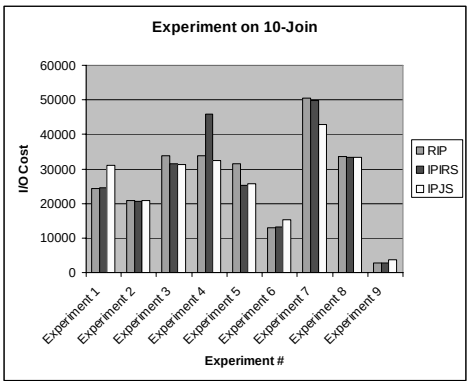


**Figure 5-5 Summary chart for 10-Join experiments**

*Experiments with 20 joins*

Amount of joins = 20

Database catalog = "Catalog1" for Experiment 1 to 3,
"Catalog2" for Experiment 4 to 6,
"Catalog3" for Experiment 7 to 9

Number of queries = 10 (taken best of them)

Population size = 100

Maximum number of generations = 500

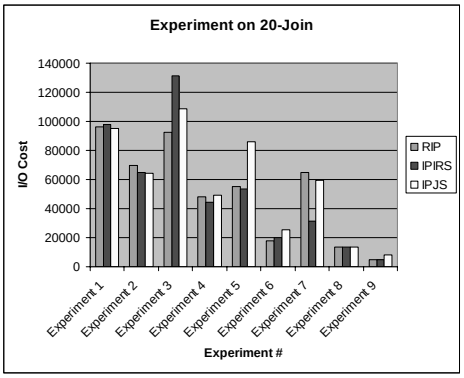


**Figure 5-6 Summary chart for 20-Join experiments**

*Experiments with 30 joins*

Amount of joins = 30

Database catalog = "Catalog1" for Experiment 1 to 3,
"Catalog2" for Experiment 4 to 6,
"Catalog3" for Experiment 7 to 9

Number of queries = 10 (taken best of them)

Population size = 100

Maximum number of generations = 500

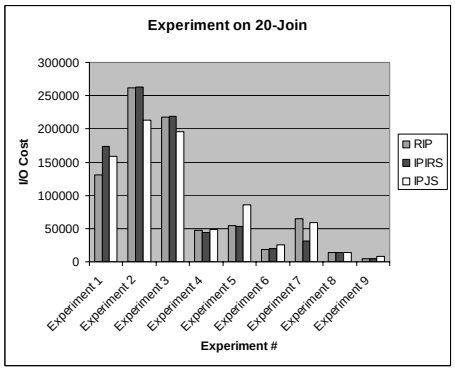


**Figure 5-7 Summary chart for 30-Join experiments**

## 6 CONCLUSIONS

In this study, heuristics based on join selectivity and the size of intermediate join result are applied before running the Genetic Programming approach on query optimization problem. This technique uses the same chromosome representation used in [STILLGER et al 96], which conforms to the principles of Genetic Programming. Heuristics are applied to the solutions space of valid query trees that represent a query graph before being executed in Genetic Programming runs.

The techniques that are using join selectivity as a heuristic criterion and the size of the intermediate join result as another heuristic criterion are compared to the ordinary Genetic Programming technique. The experiments show that the techniques that are using heuristics run more efficiently than the ordinary Genetic Programming technique in most cases. Of the two heuristic techniques, the one that uses join selectivity is slightly better than the one that uses the size of the intermediate join result.

This research was conducted using Materialization query execution method, which always creates a new temporary relation after the join operator is executed on any two relations. And the temporary relation is used as input for the next join operation. Another query execution method, Pipelining, can be used in further researches. In addition to heuristics based on join selectivity and the size of the intermediate join result, more various kind of heuristics could be applied in future experiments to exploit the possibility of more efficient heuristic criteria.

## 7 REFERENCES


[ELMASRI and NAVATHE 04] Ramez Elmasri and Shamkant B. Navathe. Fundamentals of Database Systems, Fourth Edition. Addison-Wesley, 2004.

[BANZHAF et al 97] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller and Frank D. Francone. Genetic Programming: An Introduction. Morgan Kaufmann, 1997.

[MUNTES et al 06] Victor Muntés-Mulero, Josep Aguilar-Saborit, Calisto Zuzarte, Josep-Lluis Larriba-Pey. CGO: A Sound Genetic Optimizer for Cyclic Query Graphs. International Conference on Computational Science, pages 156-163, University of Reading, UK, 2006.

[MUNTES et al 05] Victor Muntés-Mulero, Josep Aguilar-Saborit, Calisto Zuzarte, Volker Markl, and Josep Lluis Larriba. Genetic evolution in query optimization: a complete analysis of a genetic optimizer. Technical Report UPC-DAC-RR-2005-21, Dept. d'Arqu. de Computadors. Universitat Politecnica de Catalunya, 2005.

[ILYAS et al 03] Ihab F. Ilyas, Jun Rao, Guy Lohman, Dengfeng Gao and Eileen Lin. Estimating Compilation Time of a Query Optimizer. In Proceedings of ACM SIGMOD International Conference on Management of Data, San Diego, California, 2003.

[STILLGER et al 96] Michael Stillger, Myra Spiliopoulou, Johann-Christoph Freytag. Parallel Query Optimization Exploiting Bushy and Pipeline Parallelism with Genetic Programs. Institut für Informatik, Humboldt-Universität zu Berlin, Germany, 1996.

[BENNETT et al 91] Kristin Bennett, Michael C. Ferris, and Yannis Ioannidis. A Genetic Algorithm for Database Query Optimization. University of Wisconsin, Madison, WI, 1991.

[SWAMI 89] Arun Swami. Optimization of Large Join Queries: Combining heuristics and combinatorial techniques. In SIGMOD International Conference on Management of Data, pages 367-376, Portland, Oregon, 1989.

[SWAMI and GUPTA 88] Arun Swami and Anoop Gupta. Optimization of Large Join Queries. In SIGMOID International Conference on Management of Data, pages 8-17, Chicago, IL, 1988.

[KRISHNAMURTHY et al 86] Ravi Krishnamurthy, Haran Boral and Carlo Zaniolo. Optimization of Nonrecursive Queries. In Proceedings of the Twelfth International Conference on Very Large Databases, pages 128-137, Kyoto, Japan, 1986.

[IOANNIDIS and WONG 87] Yannis E. Ioannidis and Eugene Wong. Query Optimization by Simulated Annealing. In Proceedings of the 1987 ACM-SIGMOID Conference, pages 9-22, San Francisco, CA, 1987.

[IOANNIDIS and KANG 90] Yannis E. Ioannidis and Younkyung C. Kang. Randomized Algorithms for Optimizing Large Join Queries. In SIGMOID International Conference on Management of Data, pages 312-321, Atlantic City, NJ, 1990.