The Computer Journal Advance Access published January 15, 2010

doi:10.1093/comjnl/bxp130

An Evolutionary Genetic Algorithm for Optimization of Distributed Database Queries

ENDER SEVINÇ AND AHMET COŞAR*

Computer Engineering Department, Middle East Technical University, Ankara, Turkey
**Corresponding author: cosar@ceng.metu.edu.tr*

High-performance low-cost PC hardware and high-speed LAN/WAN technologies make distributed database (DDB) systems an attractive research area where query optimization and DDB design are the two important and related problems. Since dynamic programming is not feasible for optimizing queries in a DDB, we propose a new genetic algorithm (GA)-based query optimizer (new genetic algorithm (NGA)) and compare its performance with random and optimal (exhaustive) algorithms. We perform experiments on a synthetic database with replicated relations, but no horizontal or vertical fragmentation. Network links are assumed to be gigabit ethernet. Comparisons with optimal results show that our NGA formulation performs only 20% of the optimal results and we have achieved 50% improvement over a previous GA-based algorithm.

Keywords: query optimization; distributed database; genetic algorithm

Received 3 October 2009; revised 14 December 2009
Handling editor: Adnan Yazici

1. INTRODUCTION

One of the early distributed DBMSs was SDD-1 [1], which was designed for slow (64 Kbps) wide area networks and made extensive use of semi-joins to reduce communication costs. Later systems, such as R* [2, 3] and Distributed-INGRES [4], assume faster networks and did not employ semi-joins. Both R* and SDD-1 generate static unchangeable query plans, whereas Distributed-INGRES dynamically generated query execution plans (QEPs) at run-time using the information available at run-time. Neither R* nor SDD-1 considered horizontal and vertical fragments, whereas Distributed-INGRES handled only horizontal fragments. None of these three systems consider replication [5].

We are working on developing a system for designing efficient distributed databases (DDBs) with the allocation of data to distributed nodes. DDB design requires a two-step process where efficient optimization of queries is a prerequisite when searching for an optimal design [6]. We design new genetic algorithm (NGA) to generate good QEPs for a set of distributed queries and thus we are able to quickly calculate a fitness value for a given DDB schema under a given query workload [7]. Thus, we have been able to speed up the fitness value calculation and use it for evaluating DDB designs generated by a genetic algorithm (GA)-based algorithm. For this purpose, we define and incorporate a *minimal k-length block* mechanism for identifying and preserving good sub-plans in a solution. We also prove the effectiveness of our GA by comparing its performance with that of a random algorithm, as well as with optimal solutions generated by an exhaustive algorithm. We have also implemented a system to execute distributed QEPs generated by NGA on our departmental cluster machine and came up with very accurate communication cost formulas [8].

A comprehensive DDB design approach using the GA technique is presented in [9]. In [10], this GA model is extended by including network latency and considering parallel processing in response time calculations. This extended model was used to design efficient DDBs that can make use of inherent parallelism in the evaluation of DDB queries. GA is a randomized algorithm that could be run for a very long time to obtain an optimal solution. Therefore, we limit the run-time of NGA and to achieve additional improvements of GA-generated solutions we run a local optimization heuristic [11] to obtain local optimal solutions.

In Section 2, previous works using heuristic and GA-based solutions for DDB query optimization are explained. In Section

3, our NGA formulation is described. Section 4 presents the results of the experiments using a set of queries on synthetic DDB schemas. Our conclusions are presented in the last section.

2. PREVIOUS WORK

Earlier research on DDB query optimization use techniques such as dynamic programming [3, 12], simulated annealing and iterative improvement [13], sub-optimal greedy heuristics [5] and lately GA-based solutions [9, 14–17]. GA-based evolutionary solutions for the allocation of data on DDBs have also been studied [18].

2.1. Exhaustive search algorithms

System-R query optimizer is based on dynamic programming. The complete search space is constructed by permutation of the joined relations. First, all possible pairs of join relations are formed, then all possible triples and so on. This algorithm is feasible when number of joins (N) is small and there is sufficient main memory to store all calculated optimal plans. As N grows, the required memory makes the use of dynamic programming infeasible [12]. Iterative dynamic programming (IDP) [19] modifies DP by solving the query optimization problem by finding optimal plans for k-relations where DP execution will succeed using bounded memory. In the next iteration of IDP those k-relations are removed from the problem, reducing problem size to $(n - k)$. There are eight variants of IDP depending on how k-relations are chosen, such as the exploration of bushy/linear join trees and whether only one 'optimal plan' for k-relations or a set of incomparable plans is selected.

2.2. Heuristic-based query optimization

One of the best-known heuristic-based techniques used for distributed query optimization is the Distributed-INGRES algorithm derived from centralized INGRES [5]. It uses a dynamic approach making optimization decisions at run-time in addition to pre-execution time decisions. The objective function of the algorithm is to minimize a combination of both communication and execution times. Dynamic query optimization algorithm is given in [5]. In [20], an adaptive query optimization algorithm is proposed. Rather than constructing a full plan for an access path and executing it, the algorithm constructs a partial plan, executes it, updates the statistics (using the number of records returned by select and join operations) and constructs a new partial plan. Since a partial plan is constructed based on the latest statistics, the algorithm is adaptive to data modifications and less affected by statistical errors. The SDD-1 algorithm is extended by considering local processing cost as well as communication cost.

2.3. GA-based solutions

GA is a general purpose search algorithm which applies principles of natural selection to a randomly generated pool of populations consisting of chromosomes each representing a complete solution, and using these initial solutions tries to evolve better ones. For each chromosome, a fitness value is calculated to choose most competitive chromosomes that will produce the next generation. Two operators used for producing offspring are crossover and mutation [14]. In [16], GA is implemented by using two crossover encodings in order to generate left-deep and bushy join trees. The advantage of this version of GA is that it is designed for a parallel architecture, and significant computational savings over the other randomized methods can be obtained by a parallel implementation. Another three-based GA has been developed recently in [21].

In [22], GA solution to query optimization is compared with other techniques, but this work does not consider customized crossover and mutation operators. In [9], a comprehensive mathematical modelling approach is proposed for the allocation of data and operations to nodes in a computer communications network. The model makes a balanced decision between performances of retrieval and update activities issued from various nodes. It considers the concurrency control mechanism used as well as capacity constraints at nodes and on links in the network. In [23], a gene structure for DDB query optimization GA solutions is given. It consists of four parts, each corresponding to one of the four decisions in the DDB query optimization model. Selecting a replica of a relation, semi-join operations to reduce the communication cost, join site selection and join order. Table 1 shows the gene structures for two sample execution plans of a distributed query having

TABLE 1. Gene structures for QEPs [23].

Solution	Execution plan	Copy Id	Semi-join	Join-site	Join-order
1	Sample plan 1	1 3 4 4	01 10 00	0 0 4	0 2 1
2	Sample plan 2	**2 3 4 3**	**01 00 00**	**0 0 0**	**0 1 2**
3	Crossover 1,2	1 **3** 4 **3**	**01** 10 **00**	0 **0** 4	**0 2 1**
4	Mutation 3	1 3 4 4	11 10 00	1 0 4	0 2 1
5	Inversion 3	1 3 4 3	01 10 00	0 0 4	2 0 1

three join conditions in a five-node-distributed DBS containing four relations. It also illustrates the effects of genetic operators on chromosomes. The third column in Table 1, 'Copy Id' represents the site number of the chosen replica for the input files (relations).

The GA used in [15] employs uniform crossover [24] to combine file copy selections and a random mutation operator. In uniform crossover, the child inherits a value for each gene position from one or the other parent with equal probability. Solution 3 illustrates a possible result of applying the uniform crossover operator to solutions 1 and 2. The first and third file sites were taken from solution 1, the second and fourth from solution 2. In solution 4 a mutation of solution 3 is shown where BF3 is randomly selected to be mutated. The underlined mutation randomly changes its selected replica location from site S3 to site S4. Solution 5 in Table 1 illustrates a possible result of applying the inversion operator to Solution 3. The order of the first two joins is reversed from <J0, J2> to <J2, J0>. The fitness function is given below Formula (1), where cost (S) is the estimated total processing and I/O times, k is a normalizing constant [23]:

$$\text{Fitness (S)} = 1 - \frac{\text{cost (S)}}{k}. \tag{1}$$

3. AN NGA FORMULATION

Our aim is to develop a new improved GA and verify its performance by comparing it with other algorithms. The first of three algorithms that will be compared with NGA is an exhaustive search algorithm (ESA). The second algorithm randomly generates an equal number of random solutions as the NGA. If a given GA algorithm shows no improvement compared with the completely random algorithm, this shows that the mutation and crossover operators used in NGA have not made any positive contribution to the optimization process. This algorithm is called as 'Random' and shown in Section 4.1. The third algorithm is an earlier GA-based algorithm proposed in [23] and is referred as RGA in this paper. As discussed in Section 2.3, GA has a comprehensive query optimization model that integrates copy identification, join-order, join-site selection and reduction by semi-joins into a single model. It exploits the concepts of gainful semi-joins and pure join attributes. We include both network communication and local processing costs. Sites and communication links can be heterogeneous in terms of unit costs and capacities. The last algorithm is our GA-based algorithm, NGA with new mutation and crossover operators. We also use a greedy algorithm that improves a given plan by selecting copies of replicated relations at the nearest site.

3.1. Chromosome structure

All possible QEPs will be represented using a chromosome structure. Each chromosome has n genes each one corresponding to a join condition given in the query. The gene order determines in which order joins are evaluated and at which node. Execution starts with G_1 at the left most position and finishes with the last gene, G_n, at the right end.

The chromosome structure of a query is shown in Fig. 1. Each gene, G_i, has the condition number, node number and 2 bits for semi-join. Proposed crossover is named as New-Crossover and mutation as New-Mutation. There are three parameters of GA that will affect the performance of a GA-based optimization algorithm. These are (i) mutation percentage, (ii) crossover percentage and (iii) initial population size. To decide the best values for these parameters, we performed three experiments plotting performance graphics for varying values of them. The results are shown in Figs. 2–4. These results show that a

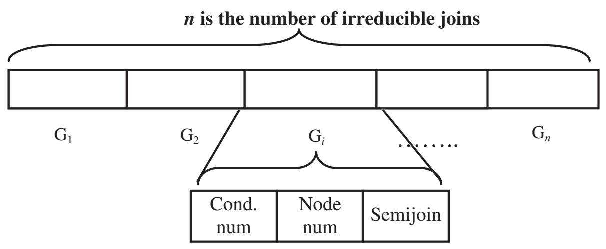

FIGURE 1. Chromosome structure.

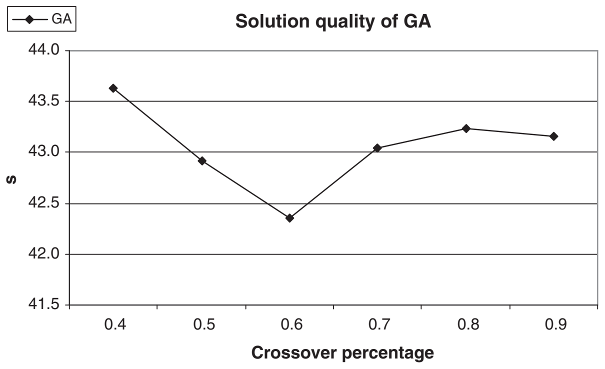

FIGURE 2. The performance of GA for increasing crossover percentages.

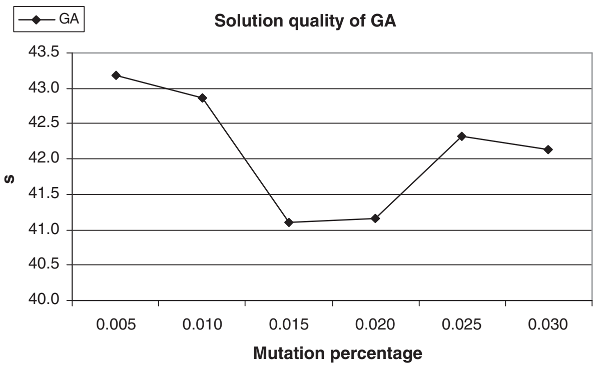

FIGURE 3. The performance of GA for increasing mutation rates.

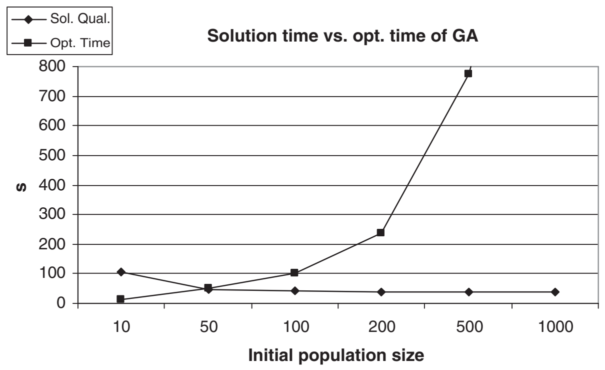

FIGURE 4. The performance of GA for increasing initial population size.

crossover percentage of 0.6, mutation rate of 0.015 and initial population size of 100 give the best results. In fact, larger population sizes will slightly improve the solutions, but only at the cost of an exponential increase in the GA run-time. We use two-point crossover with 50% truncation technique since we determined it to be better than other alternatives.

There are three methods, tournament, roulette wheel and truncate to select individuals for mating in the current solution pool. In tournament, '*k*' individuals are randomly selected and the best one among them is identified for mating. The process is repeated until required number of individuals is selected. In the roulette wheel method, all individuals are assumed to be placed on a roulette wheel. The probability of an individual to be selected is proportional to its fitness value. A random number is generated and its value is used for indexing positions on the roulette wheel choosing the individual at that position. The last selection method is truncation which simply eliminates the individuals with lowest fitness values, so that only the required individuals are left in the solution pool.

The crossover operation also has two widely used methods, one-point and two-point. In one-point, a random position is selected on the chromosome and genes up to this point are copied from the first (second) parent and remaining genes are copied from the corresponding positions of the second (first) parent. In two-point crossover, two random points are selected on the chromosome and the genes between these two points are swapped. Both one-point and two-point crossover will generate two new individuals.

To decide what combination of one-point/two-point crossover and tournament/roulette-wheel/truncate methods will give the best GA method, we implemented six combinations as defined in Table 2 and compared. The results are given in Fig. 5.

3.1.1. *New-crossover*

The number of genes for crossover is determined by multiplying the crossover ratio with the total number of genes in the chromosome. The best values are used, which are found in the previous section. In GA, usually the crossover point is decided randomly, but differently in NGA it is determined by a heuristic. This crossover heuristic has a greedy approach. It uses costs of genes for this purpose and selects a minimal cost subsequence of genes for crossover.

GA parameters are Initial Pool Size = 100, Mating Population = 50, Convergence Ratio = 95%, Crossover type = 'Truncate, 2-point', Truncate ratio = 50%, Crossover Ratio = 0.6 (60%), Mutation ratio = 0.015 (1.5%). Figures 6 and 7 describe how the chromosomes are used for New-Crossover. Since GA6 outperforms the other types of GA as seen in Fig. 5, two-point crossover will be performed and five genes out of eight will be crossed. For applying the New-Crossover

TABLE 2. Types of GAs.

GA	Selection type	Crossover type
GA1	Tournament	One-point
GA2	Tournament	Two-point
GA3	Roulette wheel	One-point
GA4	Roulette wheel	Two-point
GA5	Truncate	One-point
GA6	Truncate	Two-point

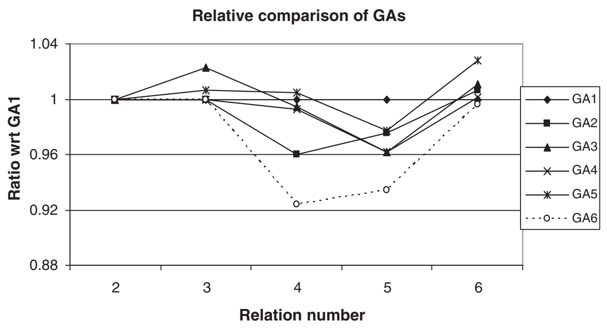

FIGURE 5. Comparison of selection and crossover solution quality.

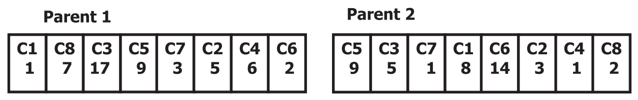

FIGURE 6. Parent chromosomes.

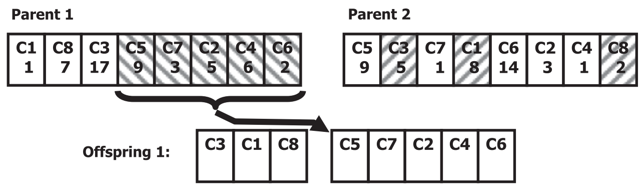

FIGURE 7. Crossover implementation (P1 × P2).

operator, the first step is to find a minimum cost subsequence of genes. For the sample chromosome, we need to find a five-gene subsequence which has a minimal total cost. In DDBS, such a minimum cost subsequence of genes will tend to use a minimal number of nodes resulting in minimal communication cost and joins with smaller input relations resulting in smaller intermediate results.

Definition 1 (minimal k-length block). *A minimum cost 'k-length' subsequence of consecutive genes is called a minimal k-length block in a chromosome and it has the lowest cost compared with all other 'k-length' subsequences of genes in that chromosome. 'k-length block size' is the size of the crossed genes between parent chromosomes and changes with respect to chromosome size and crossover ratio. In Parent1 ($P1$), we have four alternative five-length blocks. 'C1 C8 C3 C5 C7', 'C8 C3 C5 C7 C2', 'C3 C5 C7 C2 C4', 'C5 C7 C2 C4 C6'.*

'C5 C7 C2 C4 C6' has the lowest cost. The total cost of this block is 25 s (Fig. 6) and it is the smallest cost five-length gene block (in Fig. 7) in Parent P1. In Fig. 7, this minimum cost length block which includes the last five genes of P1 is taken from P1 and then put into the same gene position in the generated offspring. Then, the first three absent genes are taken from parent P2 preserving the order in which they appear in parent P2.

Definition 2 (New-Crossover). *New-Crossover is an operator which takes a minimal k-length block from the first parent and preserves the positions and orders of these genes in the generated offspring. Then, the rest of the genes are copied from the second parent in the order as they appear in it. When P1 and offspring*1 ($O1$)*, shown in Fig. 7, are compared, it is seen that only the order of the first three genes of P1 are changed. This process saves time. Crossover implementation of P1 and P2* ($P1 \times P2$) *is explained above, and* $P2 \times P1$ *is defined similarly.*

New-Mutation operator modifies any of the join node number, chosen replica and semi-join option of a gene. This operator works similar to the one given in [23]. However, the gene selection criterion is different and uses the cost of individual genes to assign proportional mutation probabilities.

Definition 3 (New-Mutation). *Costs of the genes are used as a selection probability obtained by dividing gene cost to the total chromosome cost. Then, a random number is generated; this number is used to select one of the genes for mutation where the probability of selection of a gene is proportional to its cost. The gene is randomly assigned a new node number and semi-join option. Finally, available node with the least communication cost join is selected. Table 3 shows the selection probabilities for a sample eight-gene chromosome given in Fig. 8. The third gene has the highest cost and, the first gene has the smallest cost, therefore they have the highest and lowest, respectively, probabilities for being mutated.*

TABLE 3. Selection probability of a gene in New-Mutation.

Number of gene	1	2	3	4	5	6	7	8
Cost of gene	1	7	17	9	3	5	6	2
Selection probability	0.02	0.14	0.34	0.18	0.06	0.10	0.12	0.04

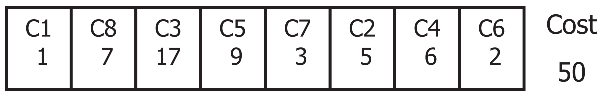

FIGURE 8. A chromosome with condition numbers and costs of the genes.

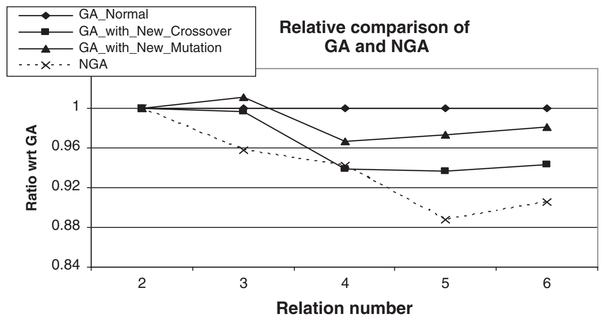

FIGURE 9. Effect of New-Crossover and New-Mutation operators on GA.

New-Mutation and New-Crossover operators are proposed to improve the performance. GA6 in Table 2 is used for this purpose. Results are shown with the increasing number of relations from two to six in a DDB environment having four nodes. This testbed is similar to the one shown in Section 4.1. We evaluated four different algorithms. The crossover and mutation operators of usual GA (GA6) are replaced with the ones in NGA resulting in two more GAs, a total of four different algorithms are run 20 times each independently and the mean of those runs is presented in Fig. 9. Semi-join operations of NGA are shown as 2 bits, 1 bit for each join relation. '00' means no semi-join operation will be performed on the input relations, while '10' and '01' represent that left and right join inputs, respectively, will be subjected to semi-join operations to reduce communication time.

3.2. Query execution model

The model is given as a graph G = (I, C, R) containing a set of join conditions (C), sites (S) and input relations/fragments (I) residing at various sites. Each condition C_i is evaluated at site S_{ki}, and the result (R_i) is sent to the next site. Fragments are shipped in parallel and the longest transfer is the communication time of Rel_i.

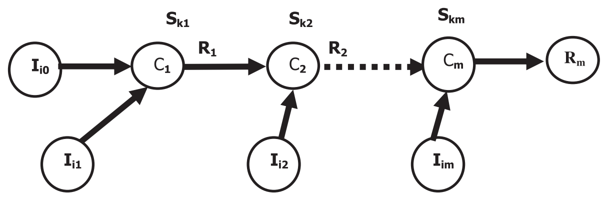

FIGURE 10. A sample QEP.

A sample QEP is given in Fig. 10. The cost of an execution plan, for n relations and m joins, denoted by Cost (P) is calculated by using Formulas (2) and (3). The selectivity factors are assigned for input relations and same values are used through all experiments.

$$\text{Cost (P)} = \sum_{i=0\cdots n} \text{comm\_cost}(\text{Rel}_i, \text{S}_{ki}) + \sum_{j=0\cdots m} \text{Proc\_cost }(\text{C}_j) \times \sum_{k=0\cdots m} \text{comm\_cost }(\text{R}_k), \quad (2)$$

$$\text{Comm\_cost }(\text{Rel}_i, \text{S}_k) = \max_{j=0\cdots \text{NF}_i} |(\text{comm\_cost }(\text{F}_{ij}, \text{S}_k), \text{ where Rel}_i \text{ has NF}_i \text{ fragments.} \quad (3)$$

4. EXPERIMENTAL SETUP AND RESULTS

To compare NGA with ESA, we used a synthetic DDB schema. An interconnection network with 1 Gbps ethernet links is assumed. Each node is assumed to be able to communicate with any other node, without considering multi-hop transmissions and store-and-forward delays. Current technology with Gbps LAN/WAN links with very low delay switching protocols (e.g. MPLS and ATM) make our assumptions realistic and low cost.

There are six defined distinct relations over attributes: BF_0(1, 2, 3, 4, 5), BF_1(6, 1, 7, 8, 9, 10), BF_2(11, 6, 12, 13, 14, 15), BF_3(16, 11, 17, 18, 19, 20), BF_4(21, 16, 22, 23, 24, 25) and BF_5(26, 21, 27, 28, 29, 30). The common attributes in these files represent the foreign keys and primary key for each file is the first attribute in the list. Since we are not comparing join methods, we do not attempt to locally optimize a given query's access paths. All queries are assumed to be irreducible. Each processing node has 102 buffers (one page), and page size is 10 240 bytes, disk I/O time is 10 ms (per page), total available buffer size for database operations is 128 MB. All processing nodes are homogenous and have the same architecture, RAM size, number of buffer pages and page sizes.

4.1. Experimental results

Five different DDB schemas have been prepared for each of the experiments. These schemas form the X-axis of the graphs. Y-axis shows the estimated costs of execution plans obtained for each schema. The results are examined by using two different scenarios. First, the number of nodes is increased from two to six while the same linear type query which refers to four relations is used for each case, and results are given in Fig. 11. In Fig. 11a, ESA gives the optimum value and helps us evaluate the performance of GA algorithms by comparing how close it is to the optimum value. We also give results for a 'Random' algorithm which generates the same number of solutions as the GA algorithms and we use it to determine if the GA contributes positively. If a GA performs no better than the 'Random' algorithm, we can conclude that its crossover/mutation operators and selection strategy are ineffective. Our experiments showed that NGA finds acceptable solutions compared with ESA and GA. When the problem size increases, the time needed for ESA grows exponentially as in Fig. 11b, making ESA too costly and NGA a very competitive and cheap alternative.

Figure 12 presents the effects of increasing 'number of relations' on performance. Figure 12a shows the 'solution quality', the execution time of the query plan to answer the query. When the problem size is small, all algorithms produce very close cost solutions. However, as the problem size increases, ESA produces 10% to 15% better solutions than NGA. In Fig. 12b, it is seen that as problem size grows with the number of relations, the optimization time of the ESA algorithm increases exponentially. Thus, as problem size grows, NGA becomes a better alternative than ESA and GA.

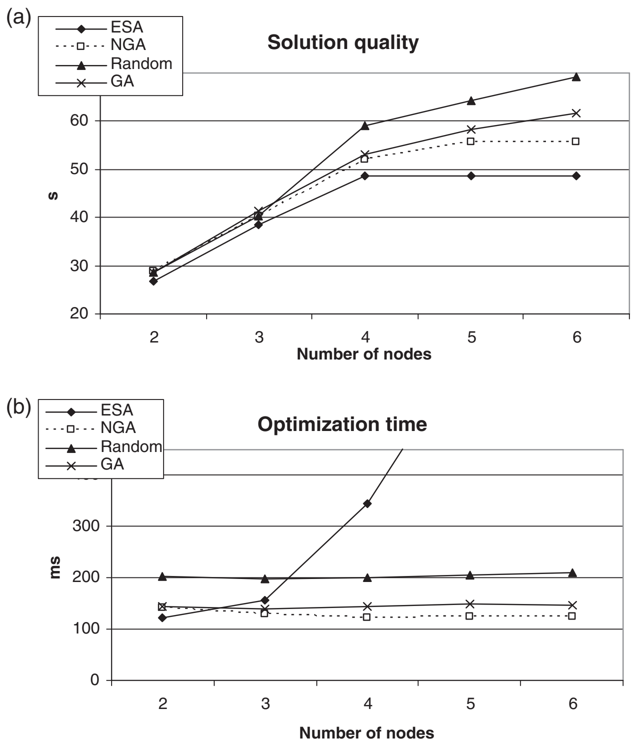

FIGURE 11. The effect of increasing number of nodes.

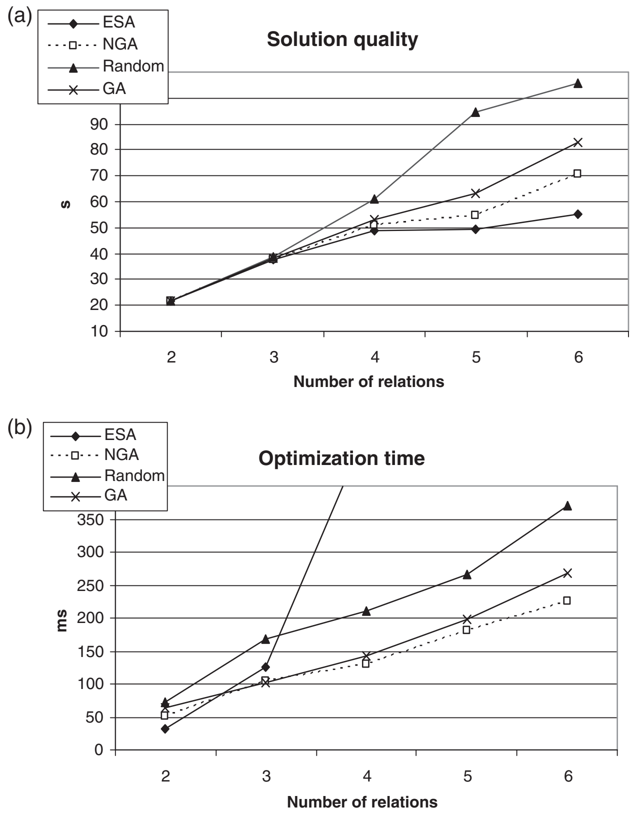

FIGURE 12. The effect of increasing number of relations.

5. EXECUTION ON CLUSTER MACHINE

In this section, we study the performance of our distributed QEPs in our department's cluster [25] computer system. The nodes in the cluster are connected using a high-performance switch which allows all nodes to directly communicate with all other cluster nodes with the same performance. The QEP is loaded by node0 and related parts of the query plan tree are sent to processing nodes telling each node which relation(s) they should input and which site should receive the output [8]. We do not consider the problems of relation fragmentation (horizontal or vertical [26]) and replication of relations to improve performance [27] by increasing parallel processing.

In order to be able to efficiently use the communication and processing capabilities of the cluster system, we first experimentally determined the optimal message size to be used for transferring relation data among cluster nodes. For this purpose, a relation transfer was performed among cluster nodes using message sizes of 4–1000 bytes. The results of this experiment are given in Fig. 13 and show that there is a big improvement until message data size reaches 40 bytes (probably due to minimum ethernet packet size of 64 bytes) and then there are slight improvements until message data size reaches 400 bytes and there is very small improvement for up to 1000 bytes. We conclude that transferring a relation between nodes using message data sizes of 400–1000 bytes would give good performance.

We calculate two communication parameters from this experiment, *overhead per message* and *cost per byte*. The first parameter is (9 s/10E6 messages), which is about 0.9 μs. The second parameter is 0.2 s/40 × 1E6 bytes, which is 0.005 μs for a 1-byte message. The final communication time formula is:

$$\text{No. of messages} * (0.9\,\mu\text{ s} + 0.005\,\mu\text{ s} * \text{BytesPerMsg})$$

Using 4-byte messages, we need 10 × 1E6 messages, and the above formula gives 10E6 * (0.9 μs + 0.005 μs * 4) = 9.2 s. Using 1000-byte messages, we need 40 000 messages and the formula gives 40 000 * (0.9 μs + 0.005 μs * 1000) = 40 000 * 5.9 μs = 0.236 s. These calculated results perfectly match both end points of the line in Fig. 13. To show the effect of message size on actual execution time of a distributed query plan, we execute the plan given in Fig. 14 using our departmental cluster machine with increasing message sizes and report timing results in Fig. 15.

The distributed plan is given in Fig. 14. Nodes with 'Jx' represent join nodes, and leaf nodes represent the input base relations. The 'Nx' in each box represents the node where the

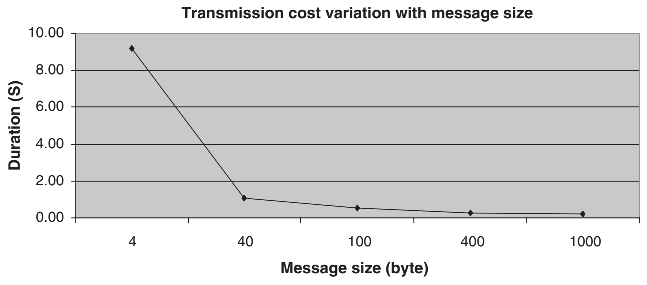

FIGURE 13. Optimal message size for data transfer in cluster system.

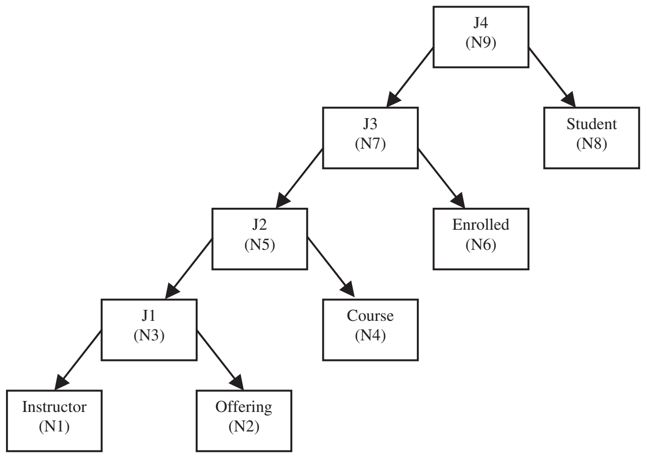

FIGURE 14. A sample distributed QEP.

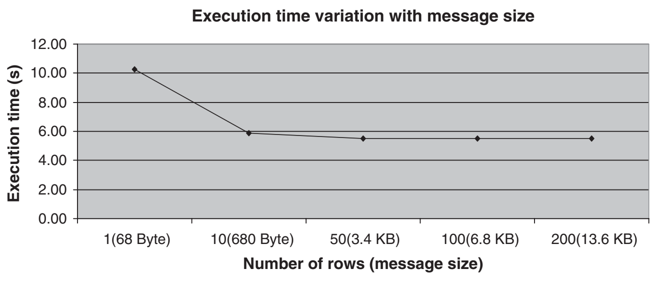

FIGURE 15. Execution time of a DDB query as message size is increased.

relation is for leaf nodes and the node where join operation will be executed for non-leaf nodes.

6. CONCLUSIONS AND FUTURE WORK

In this work, a set of alternatives for a GA-based solution to the DDB query optimization problem is evaluated. We compared the performance of our NGA with a previously defined GA. We also implemented ESA and random algorithms to obtain lower and upper bounds, respectively, in our experiments. We showed that our NGA produced much better results than a random search algorithm, proving that it was useful and we were able to produce solutions about 10–15% worse than the best achievable solution (obtained using ESA) while NGA optimization times are much lower than ESA. The reason NGA has performed better than other GA algorithms is its ability to locally improve solutions by selecting the nearest copy of replicated relations. Its capability to identify and preserve low-cost good sub-plans by encouraging selection of minimal cost k-length gene blocks to be inherited by offspring. Our NGA also directs the search towards more expensive parts of a chromosome by assigning higher selection probabilities to more expensive genes.

We also investigated the effect of increasing number of nodes and number of relations on the performance of NGA and showed that NGA continues to produce good results as problem size grows. To verify and improve our cost formulas for a cluster computer, experiments were performed to measure the effect of message sizes on distributed query execution times. We verified our communication cost formulas by executing an actual DDB query plan on a synthetically generated relational database.

As future work, we are planning to extend our exhaustive algorithm to work on a cluster machine and thus be able to compare optimal results with our NGA for 20+ node DDB with a comparable number of relations. Development of algorithms for query optimization on DDB with heterogeneous LAN/WAN links and bandwidths especially in the domain of data warehouse applications would also be a very interesting research direction as such systems are becoming more widespread with low-cost, high-bandwidth WAN links.

REFERENCES

[1] Apers, P.M.G., Hevner, A.R. and Yao, S.B. (1983) Optimization algorithms for distributed queries. *IEEE Trans. Softw. Eng.*, **9**, 57–68.

[2] Lohman, G., Mohan, C., Haas, L., Lindsay, B., Selinger, P., Wilms. P. and Daniels, D. (1985) Query Processing in R*. In Kim W., Batory D. and Reiner D. (eds.), *Query Processing in Database Systems*. Springer.

[3] Selinger, P.G. and Adiba, M.(1980) Access Path Selection in Distributed Database Management Systems. *Proc. First Int. Conf. Data Bases*, pp.204–215.

[4] Epstein, R., Stonebraker, M. and Wong, E. (1978) Query Processing in a Distributed Relational Database System. *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 169–180.

[5] Ozsu, M.T. and Valduriez, P. (1999) *Principles of Distributed Database Systems* (2nd edn). Prentice Hall.

[6] Papadomanolakis, S., Dash, D. and Ailamaki, A. (2007) Efficient Use of the Query Optimizer for Automated Database Design. *VLDB'07*, pp. 23–28.

[7] Bayir, M., Toroslu, I.H. and Cosar, A. (2007) Genetic algorithm for the multiple-query optimization problem. *IEEE Trans. Syst. Man Cybern. C Appl. Rev.*, **37**, 147–153.

[8] Onder, I.S. (2010) Execution of distributed database queries on a HPC system. MS Thesis, Department of Computer Engineering, Middle East Technical University, Ankara, Turkey.

[9] March, S.T. and Rho, S. (1995) Allocating data and operations to nodes in a distributed database design. *IEEE Trans. Knowl. Data Eng.*, **7**, 305–317.

[10] Johansson, J.M., March, S.T. and Naumann, J.D. (2003) Modeling network latency and parallel processing in DDB design. *Decis. Sci.*, **34**, 677–706.

[11] Gen, M. and Cheng, R. (2000) *Genetic Algorithms and Engineering Optimization*. Wiley, New York.

[12] Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A. and Price, T.G. (1979) Access Path Selection in a Relational Data Base System. *Proc. ACM-SIGMOD Conf. Management of Data*, pp. 23–34.

[13] Ioannidis, Y.E. and Kang, Y.C. (1990) Randomized Algorithms for Optimizing Large Join Queries. *Proc. ACM SIGMOD Int. Management of Data*, USA, May, pp. 312–321.

[14] Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing.

[15] March, S.T. and Rho, S. (1994) A Nested Genetic Algorithm for Distributed Database Design. *Proc. 27th Annual Hawaii Int. Conf. System Sciences*.

[16] Bennett, K., Ferris, M.C. and Ioannidis, Y.E. (1991) A Genetic Algorithm for Database Query Optimization. *Proc. 4th Int. Conf. Genetic Algorithms*.

[17] Dong, H. and Liang, Y. (2007) Genetic Algorithms for Large Join Query Optimization. *Genetic and Evolutionary Computing Conf.*, pp. 1211–1218.

[18] Ahmad, I., Karlapalem, K., Kwok, Y. and So, S. (2002) Evolutionary algorithms for allocating data in distributed database systems. *Distrib. Parallel Databases*, **11**, 5–32.

[19] Kossmann, D. and Stocker, K. (2000) Iterative dynamic programming: a new class of query optimization algorithms. *ACM Trans. Database Syst.*, **25**, 43–82.

[20] Yu, M.J. and Sheu, P.C.-Y. (1997) Adaptive join algorithms in dynamic distributed databases. *Distrib. Parallel Databases*, **5**, 5–30.

[21] Li, H. and Luo, B. (2008) A Tree-based Genetic Algorithm for Distributed Database. *Proc. IEEE Int. Conf. Automation and Logistics*, Qingdao, China, pp. 2614–2618.

[22] Zhou, Z. (2007) Using heuristics and genetic algorithms for large-scale database query optimization. *J. Inf. Comput. Sci.*, **2**, 261–280.

[23] Rho, S. and March, S.T. (1997) Optimizing distributed join queries: a genetic algorithm approach. *Ann. Oper. Res.*, **71**, 199–228.

[24] Syswerda, G. (1989) Uniform Crossover in Genetic Algorithm. *Proc. 3rd Int. Conf. Genetic Algorithms*, pp. 2–9.

[25] Mach, W. and Schikuta, E. (2009) Parallel algorithms for the execution of relational database operations revisited on grids. *Int. J. High Perform. Comput. Appl.*, **23**, 152–170.

[26] Song, S. and Gorla, N. (2000) A genetic algorithm for vertical fragmentation and access path selection. *Comput. J.*, **43**, 81–93.

[27] Chang, R. and Chen, P. (2007) Complete and fragmented replica selection and retrieval in data grids. *Future Generation Comput. Syst.*, **23**, 536–546.