International Journal of Computer Theory and Engineering, Vol.3, No.1, February, 2011
1793-8201

Distributed Query Processing Plans Generation using Genetic Algorithm

T.V. Vijay Kumar, Vikram Singh and Ajay Kumar Verma

Abstract—**Large amount of information available in distributed databases needs to be exploited by organizations in order to be competitive in the market. In order to exploit this information, queries are posed thereupon. These queries require efficient processing, which mandates devising of optimal query processing strategies that generate efficient query processing plans for a given distributed query. The number of possible query processing plans grows rapidly with increase in the number of sites used, and relations accessed, by the query. There is a need to generate efficient query processing plans from among all possible query plans. The proposed approach attempts to generate such query processing plans using genetic algorithm. The approach generates query plans based on the closeness of data required to answer the user query. The query plans having the required data residing in fewer sites, are considered more efficient, and are thus preferred, over query plans having data spread across a large number of sites. The query plans so generated involve minimum number of sites for answering the user query leading to efficient query processing. Further, experimental results show that the GA based approach converges quickly towards the optimal query processing plans for an observed crossover and mutation rate.**

Index Terms—**Distributed Query Processing, Genetic Algorithm**

I. Introduction

A distributed database (DDB) is a collection of data which is logically interrelated and distributed over various sites of a computer network [5], [23]. A system that manages such distributed database is the distributed database system (DDBS)[23]. The performance of a DDBS is determined by its ability to process queries in an efficient manner [24], [21]. Query processing in DDBS [17], [18] requires transmission of data between sites in the network [10]. The major cost incurred while processing queries in DDBS are the CPU, I/O and communication cost, of which the communication cost is considered as the most important factor impacting the cost of query processing [16], [23]. The communication cost is the cost of transmission of data, among the participating database sites, for answering the query [23], [30]. The data transmissions, along with the local data processing, constitute a distribution strategy for a query. This strategy is referred to as Distributed Query Processing (DQP) [2], [4], [6], [19], [26], [30].

In DDBS [21], the query submitted by the user is first parsed whereafter an effective query processing plan is devised for it. The sub-queries produced for individual sites, using the plan, are processed in parallel at these sites. Results from these are then combined to produce the final integrated result. DQP aims to transform a user query into an efficient query processing strategy [3], [20], [23]. That is, DQP devises an optimal query processing strategy generating query processing plans, which reduce the amount of data transfer among sites in order to improve the response time of user queries [30]. The response time, and total time, can be used to measure the quality of DQP strategies [10].

Most of the queries on distributed relational databases require access to relations from multiple sites for their processing. The number of possible alternative query plans increases exponentially with increase in the number of relations required for processing the query [14]. The query optimizer needs to explore the large search space for generating optimal query plans. This problem becomes more complex when the required data is replicated across multiple sites whereupon there can be a large number of possible query plans for answering the user query. This number would increase with increase in the number of sites containing the required data. As a result, there is a further increase in the size of the search space of alternative query plans making query plan generation a much more complex task. Exploring all the query plans in this large search space, i.e. exhaustive search, is not feasible [14]. This problem in distributed databases is a combinatorial optimization problem [15] and has been addressed by techniques like simulated annealing, iterative improvement, two-phase optimization, etc [12], [14], [27]. These techniques, which reduce the search space, are based on plan transformation and have a cost model to assess the quality of query processing plans. However, efficiency of these techniques is affected by the unconventional behavior, in specific instances, of the problem [13]. The evolutionary computation has been found to be viable approach and Genetic Algorithm has been considered for query optimization [1], [8], [9],[28].

In this paper, an approach that generates query plans, based on closeness of data required to answer the user query, is proposed. The closeness of data is based on the number of sites involved in a query plan, whereupon the query plan involving fewer sites is considered ‘close’ and is therefore generated before the query plan involving large number of

T.V. Vijay Kumar is presently an Assistant Professor at School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, India (Email: tvvijaykumar@hotmail.com).

Vikram Singh is presently an Assistant Manager, AG3 Department, IT Division at Maruti Suzuki India Limited, Gurgaon, Haryana, India (Email: vikramsdream@gmail.com).

Ajay Kumar Verma is presently pursuing his Ph.D. from School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, India (Email: ajayverma81@gmail.com).

sites. As a result, the query plan that is generated first would involve the least number of sites enabling efficient query processing. The rationale behind the approach is that query processing over lesser sites would be more efficient.

A. Motivation

In distributed databases, data may be replicated at various sites spread over the computer network. As a result, there can be many possible combinations of relations, which may be able to provide answers to a user query. Each such combination can be said to constitute a query plan. The query plan that results in efficient query processing is the most preferred and should be generated for a given query. As an example consider the following user query

Select ProjectName, PartName,
SupplierName, DepartmentName
From Project, Part, Supplier, Supply
Where Project.ProjNo=Supply.ProjNo and
Part.PartNo=Supply.PartNo and
Supplier.SupplierNo=Supply.Supplierno

Project, Part, Supplier and Supply are the relations accessed by the query. Since the number of relations accessed by the query is 4, length of the query plan is 4. The content of the query plans are the sites containing the relations in the FROM clause. Suppose relation Project is at site (S1, S4, S6, S9), relation Part is at site (S1, S4, S5, S7), relation Supplier is at site (S1, S3, S5, S8) and relation Supply at site (S1, S2, S4, S7). The relations and their corresponding sites are given in Figure 1.

Relation	Sites			
Project	S2	S5	S7	S9
Part	S2	S5	S6	S8
Supplier	S2	S4	S6	S9
Supply	S2	S3	S5	S8

Figure 1. Relations along with their sites

The valid query plans are given in Figure 3.

7	5	2	2	Project in site S7, Part in site S5, Supplier in site S2 and Supply in site S2
9	6	4	2	Project in site S9, Part in site S6, Supplier in site S4 and Supply in site S2
2	2	2	3	Project in site S2, Part in site S2, Supplier in site S2 and Supply in site S3
2	2	2	2	Project in site S2, Part in site S2, Supplier in site S2 and Supply in site S2

Figure 2. Query Plans

The number of possible valid query plans, which is computed as ($NS_1 \times NS_2 \times NS_3 \times NS_4$), where NS_i is the number of sites containing the i^{th} relation, is 256(4×4×4×4). This number will grow with increase in the total number of sites containing the relations in the FROM clause of the user query. Accordingly, there is a need to generate query plans that would result in efficient query processing.

One way to achieve efficient query processing is by reducing the number of sites involved in query processing. Lesser the number of such sites, lesser would be the numbers of site-to-site communications. This, in-turn, may improve the efficiency of query processing. Accordingly, the query plans should ideally involve lesser number of sites. For the query plans given in Figure 2, the first query plan involves 3 sites, the second query plan involves 4 sites, the third and fifth query plans involve 2 sites each and the fourth query plan involves 1 site. Accordingly, the fourth query plan is preferred over the others as it involves the least number of sites i.e. 1.

Another important aspect is that there can be many query plans involving the least number of sites. In this case, it would be preferable if most of the joins between relations exist in a single site i.e. query plans having sites with higher concentration of relations. For the query plans in Figure 2, the third and the fifth query plans involve the same number of sites, i.e. 2. The third query plan has three relations Project, Part and Supplier residing in site S2 and relation Supply at site S3 whereas the fifth query plan has two relations each in site S5 and in site S2. This implies that the third query plan has a higher concentration of relations at an individual site i.e. 3 and therefore should be preferred over the fifth query plan.

The above two aspects are used to define a 'close' query plan. The query plan involving fewer sites, and having higher concentration of relations, is considered more 'close' and is preferred over the others. For the query plans in Figure 2, the ordering of query plans, based on descending order of closeness, is given in Figure 3.

2	2	2	2
2	2	2	3
5	5	2	2
7	5	2	2
9	6	4	2

Figure 3. Query Plans ordered based on 'closeness'

The query plans higher in the order involve fewer sites and therefore should be generated before query plans that are lower in the order and involving larger number of sites.

This paper proposes an approach that uses Genetic Algorithm (GA) to generate 'close' query plans. The approach aims to generate query plans that are optimal with respect to the number of sites involved, and the concentration of relations in these sites, for answering the user query. This in turn would result in efficient query processing. This paper is an extended version of [29].

The paper is organized as follows: Section II discusses the approach for query plan generation followed by an example illustrating the approach in section III. The experimental results are given in section IV. Section V is the conclusion.

II. Query Plan Generation

As the data required to process the user query is spread over various sites, there is a need to arrive at a query processing plan that entails an optimal cost for query

processing. The approach presented in this paper attempts to generate such query plans for an SQL query posed on databases distributed across various nodes in the network. In distributed database, the efficiency of query processing depends upon how close the required data resides. The closer the required data resides, more efficient will be the query processing. The proposed approach aims to generate query plans with the required data residing close to each other.

Many query plans can be generated for a given query, as relations accessed by them may have multiple copies residing at various sites. As a result, there can be various combinations of relations/sites that can be used for query processing. This may lead to generation of a large number of query plans. Among these query plans, there is a need to identify optimal query plans having the required relations residing close to one other. This optimal query plan generation becomes a complex problem if the number of relations accessed by the query is high in number and each of these relations has multiple copies spread across various sites over the network. In this paper, an attempt has been made to generate such optimal query plans for a given query using the genetic algorithm. In this regard, two heuristics have been defined and are discussed next.

A. The Two Heuristics

The proposed approach uses two heuristics to generate query plans for a given query. The first heuristic is based on the number of sites required to process a user query. The second heuristic is the concentration of the relations, accessed by the query in the individual participating sites. The rationale behind the first heuristic is that, lesser the number of sites involved in query processing, lesser will be the communication between the sites. As a result, query processing will be efficient. Further, if there is more than one query plan having the minimum number of required sites, the query plan having sites with greater concentration of relations provides efficient results and shall accordingly be preferred over the others. Based on the two heuristics, a cost function, that computes the cost of proximity of data relevant for answering a user query, is defined. This cost, referred to as Query Proximity Cost (QPC), is given below:

$$QPC = \sum_{i=1}^{M} \frac{S_i}{N}\left(1 - \frac{S_i}{N}\right)$$

where M is the number of sites accessed by the query plan , S_i is the number of times the i^{th} site is used in the Query plan and N is the number of relations accessed by the query. The QPC varies from 0 to (N-1)/N. Zero indicates that all the relations accessed by the queries, reside in the same site and therefore will be the closest. On the other hand, (N-1)/N specifies that each of the relations, accessed by the query, resides in different sites and therefore are the least closest. The query plans having less QPC are generated using the genetic algorithm, which is discussed next.

B. Genetic Algorithm

Genetic algorithm (GA) is a search and optimization algorithm that follows the natural evolutionary process according to which living organisms adapt themselves to changes in the environment [22]. GA consists of encoding schemes, fitness function, and selection of parent's, genetic operators (crossover, mutation and inversion) [11]. GA starts with the population of chromosomes, which can be encoded using binary strings, real numbers, permutations of elements, lists of rules, program rules [22]. The fitness value of each chromosome in the population, using the fitness function, is evaluated. Designing the Fitness function is one of the most crucial aspects of GA as it reflects the "fitness" or "figure of merits" of a chromosome or population indicating the ability or utility of that chromosome [7], [11], [22]. The fitter individuals are then selected for crossover and mutation to arrive at the population for the next generation. GA explores the entire solution space to arrive at an optimal set of chromosomes. The proposed approach models the query processing plan generation problem as a single-objective genetic algorithm where the objective is to generate query plans with minimum QPC. The proposed approach is given next.

C. The Proposed Approach

The proposed approach generates query plans based on the closeness of the data required for answering the query. The algorithm based on this approach is given in Figure 4. The algorithm takes the relations in the FROM clause, the sites containing these relations, the probability of crossover, probability of mutation and the pre-specified number of generations as input, and produces the top-K query plans as output.

Input: RS : Relations in the FROM clause of the user query along with the sites containing them
P_c : Probability of Crossover,
P_m : Probability of Mutation,
G_P : Pre-specified number of generations

Output: Top-K query plans

Method:
Generation=0
Step 1: Generate initial population of query plans P_{QP} conforming to RS
Step 2: Evaluate QPC of each query plan based on closeness heuristics

$$QPC = \sum_{i=1}^{M} \frac{S_i}{N}\left(1 - \frac{S_i}{N}\right)$$

where
M is the number of sites accessed by the query plan,
S_i is the number of times the i^{th} site is used in the query plan,
N is the number of relations accessed by the query
Step 3: IF (Generation ≥ G_P)
Return Top-K query plans
Step 4: Select query plans using unbiased tournament selection method
Step 5: Apply crossover with Crossover probability (P_c)
Step 6: Apply mutation with mutation probability (P_m)
Step 7: Generation = Generation + 1
Step 8: GO TO **Step 2**

Figure 4. Query Plan Generation Algorithm

Using GA, the approach first generates a population of chromosomes, where each chromosome represents a query plan. The approach takes into consideration two aspects namely, the content of the query plans and the length of the query plans. The length of a query plan is computed as the number of relations in the FROM clause of the user query. The content of a query plan is the sites containing these relations.

For the three query plans given in the Figure 2, the QPC is

given in Figure 5.

SNo	Query plans	$QPC = \sum_{i=1}^{M} \frac{S_i}{N}\left(1 - \frac{S_i}{N}\right)$	QPC
1	[7,5,2,2]	1/4(1-1/4)+1/4(1-1/4)+2/4(1-2/4)	5/8
2	[9,6,4,2]	1/4(1-1/4)+1/4(1-1/4)+ 1/4(1-1/4)+1/4(1-1/4)	3/4
3	[2,2,2,3]	3/4(1-3/4)+1/4(1-1/4))	3/8
4	[2,2,2,2]	4/4(1-4/4)	0
5	[5,5,2,2]	2/4(1-2/4)+2/4(1-2/4)	1/2

Figure 5. Query Plans and their QPC

From Figure 5, query plan 4 is the most optimal followed by query plan 3. Query plan 2 is the least optimal among the five query plans.

Using the QPC values of the query plans, fitter query plans are selected using the unbiased tournament selection technique [25]. The selected query plans undergo crossover, with probability P_c, and mutation, with probability P_m, to generate the population for the next generation. This continues until the algorithm runs for a pre-specified number of generations G_P. The top-query plans are then generated based on the QPC values.

An example is given next that illustrates the proposed approach.

III. An Example

Let us consider a user query that accesses eight relations R1, R2, R3, R4, R5, R6, R7 and R8. Suppose top query plans are required to be generated for this user query

First, sites storing the relations accessed by the query are identified. These are given in Figure 6.

Relations	Sites									
R1	S1	S2	S3	S5	S6	S8	S10	S15		
R2	S1	S2	S5	S7	S8	S11	S12	S16		
R3	S1	S2	S5	S7	S8	S10	S11	S13	S15	S16
R4	S1	S2	S7	S8	S11	S14	S15			
R5	S1	S2	S7	S15	S16					
R6	S1	S3	S4	S6	S7	S11	S15			
R7	S1	S5	S6	S8	S11	S16				
R8	S1	S3	S8	S9	S12	S14				

Figure 6. Relations along with their sites

The proposed approach uses this information about the sites to generate the top-k query plans using the genetic algorithm. Suppose the initial population consist of 20 query plans. Let these 20 query plans be the ones shown in Figure 7.

[1,1,2,2,2,3,5,3]
[3,5,7,8,15,4,6,8]
[6,7,8,11,16,6,8,9]
[10,11,11,15,16,11,16,14]
[8,8,10,14,16,11,11,14]
[15,12,13,11,15,15,11,12]
[1,2,5,7,2,4,6,8]
[3,5,7,8,15,3,5,3]
[2,2,2,2,2,3,5,3]
[2,1,13,2,15,3,5,3]
[8,8,16,14,16,15,16,14]
[3,7,7,8,16,7,16,3]
[2,2,2,2,15,15,16,14]
[1,1,8,8,2,7,11,14]
[8,8,8,8,2,7,8,8]
[5,16,15,15,16,6,8,9]
[1,1,1,1,15,15,16,14]
[10,11,11,8,7,3,5,3]
[15,16,15,15,15,15,16,14]
[1,1,8,8,1,7,8,8]

Figure 7. Query Plans

The fitness of the 20 query plans are computed using the fitness function QPC given in Figure 1. The query plans and their QPC values are given in Figure 8.

SNo	Query Plan	$QPC=\sum_{i=1}^{M} \frac{S_i}{N}\left(1-\frac{S_i}{N}\right)$	QPC
1	[1,1,2,2,2,3,5,3]	[3/8*(1-3/8)+ 2/8*(1-2/8)+2/8*(1-2/8)+ 1/8*(1-1/8)]	46/64
2	[3,5,7,8,15,4,6,8]	[2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)+1/8*(1-1/8)+ 1/8*(1-1/8)+1/8*(1-1/8) +1/8*(1-1/8)]	54/64
3	[6,7,8,11,16,6,8,9]	[2/8*(1-2/8)+2/8*(1-2/8)+1/8*(1-1/8)+ 1/8*(1-1/8)+ 1/8*(1-1/8)+1/8*(1-1/8)]	52/64
4	[10,11,11,15,16,11,16,14]	[3/8*(1-3/8)+2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)+ 1/8*(1-1/8)]	48/64
5	[8,8,10,14,16,11,11,14]	[2/8*(1-2/8)+2/8*(1-2/8)+2/8*(1-2/8)+1/8(1-1/8)+ 1/8*(1-1/8)]	50/64
6	[15,12,13,11,15,15,11,12]	[3/8*(1-3/8)+2/8*(1-2/8)+2/8*(1-2/8)+1/8*(1-1/8)]	46/64
7	[1,2,5,7,2,4,6,8]	[2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)+1/8*(1-1/8)+ 1/8*(1-1/8)+1/8*(1-1/8)+1/8*(1-1/8)]	54/64
8	[3,5,7,8,15,3,5,3]	[3/8*(1-3/8)+2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)+ 1/8*(1-1/8)]	48/64
9	[2,2,2,2,2,3,5,3]	[5/8*(1-5/8)+2/8*(1-2/8)+1/8*(1-1/8)]	34/64
10	[2,1,13,2,15,3,5,3]	[2/8*(1-2/8)+2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)+ 1/8*(1-1/8) +1/8*(1-1/8)]	52/64
11	[8,8,16,14,16,15,16,14]	[3/8*(1-3/8)+2/8*(1-2/8)+2/8*(1-2/8)+1/8*(1-1/8)]	46/64
12	[3,7,7,8,16,7,16,3]	[3/8*(1-3/8)+2/8*(1-2/8)+2/8*(1-2/8)+1/8*(1-1/8)]	46/64
13	[2,2,2,2,15,15,16,14]	[4/8*(1-4/8)+2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)]	42/64
14	[1,1,8,8,2,7,11,14]	[2/8*(1-2/8)+2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)+ 1/8*(1-1/8)+1/8*(1-1/8)]	52/64
15	[8,8,8,8,2,7,8,8]	[6/8*(1-6/8)+1/8*(1-1/8)+1/8*(1-1/8)]	26/64
16	[5,16,15,15,16,6,8,9]	[2/8*(1-2/8)+2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)+ 1/8*(1-1/8)+1/8*(1-1/8)]	52/64
17	[1,1,1,1,15,15,16,14]	[4/8*(1-4/8)+2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)]	42/64
18	[10,11,11,8,7,3,5,3]	[2/8*(1-2/8)+2/8*(1-2/8)+1/8*(1-1/8)+1/8*(1-1/8)+ 1/8*(1-1/8)+1/8*(1-1/8)]	52/64
19	[15,16,15,15,15,15,16,14]	[5/8*(1-5/8)+2/8*(1-2/8)+1/8*(1-1/8)]	34/64
20	[1,1,8,8,1,7,8,8]	[4/8*(1-4/8)+3/8*(1-3/8)+1/8*(1-1/8)]	38/64

Figure 8. Query Plans

The unbiased tournament selection is then used to find fitter query plans. This selection process is given in Figure 9.

A fitter individual is characterized by a lower value of QPC. The selected query plans are given in Figure 10.

i	p(i)	QPC(i)	QPC(p(i))	I(i)
1	19	46/64	34/64	19
2	16	54/64	52/64	16
3	13	52/64	42/64	13
4	12	48/64	46/64	12
5	8	50/64	48/64	8
6	17	46/64	42/64	17
7	10	54/64	45/64	10
8	9	48/64	34/64	9
9	11	34/64	46/64	9
10	3	45/64	52/64	10
11	7	46/64	54.64	11
12	4	46/64	48/64	12
13	2	42/64	54/64	13
14	20	52/64	38/64	20
15	14	26/64	52/64	15
16	18	52/64	52/64	16
17	1	42/64	46/64	17
18	6	52/64	46/64	6
19	5	34/64	50/64	19
20	15	38/64	26/64	15

Figure 9. Unbiased Tournament Selection Process

SNo	Query plans	QPC
6	[15,12,13,11,15,15,11,12]	46/64
8	[3,5,7,8,15,3,5,3]	48/64
9	[2,2,2,2,2,3,5,3]	34/64
10	[2,1,13,2,15,3,5,3]	45/64
11	[8,8,16,14,16,15,16,14]	46/64
12	[3,7,7,8,16,7,16,3]	46/64
13	[2,2,2,2,15,15,16,14]	42/64
15	[8,8,8,8,2,7,8,8]	26/64
16	[5,16,15,15,16,6,8,9]	52/64
17	[1,1,1,1,15,15,16,14]	42/64
19	[15,16,15,15,15,15,16,14]	34/64
20	[1,1,8,8,1,7,8,8]	38/64

Figure 10. Selected Query Plans

If P_c= 0.6 then 8 query plans, out of the 12 selected query plans, are picked randomly for crossover. Suppose these query plans are as given in Figure 11.

SNo	Query plans
11	[8,8,16,14,16,15,16,14]
15	[8,8,8,8,2,7,8,8]
8	[3,5,7,8,15,3,5,3]
20	[1,1,8,8,1,7,8,8]
17	[1,1,1,1,15,15,16,14]
10	[2,1,13,2,15,3,5,3]
19	[15,16,15,15,15,15,16,14
9	[2,2,2,2,2,3,5,3]

Figure 11. Query Plans picked for Crossover

These query plans undergo crossover, followed by mutation, to generate query plans for the next generation. This process of selection, crossover and mutation are carried out for a pre-specified number of generations whereafter the top-k query plans are generated as output.

The top-k query plans, so generated using the proposed approach, would have lower value of QPC i.e. the top-k query plans would involve lesser number of sites for processing a user query. As a result, query processing would require less site-to-site communication and shall thereby result in efficient query processing. This in turn would facilitate the decision making process.

In order to ascertain the quality of query plans, i.e. the minimum average QPC values that can be attained, using the proposed genetic algorithm based approach, experiments were performed. The results and observations based on these experiments are discussed next.

IV. Experimental Results

The GA based algorithm is implemented in MATLAB 7.4 in a Windows XP environment, with Intel 1.6 GHz PC having 2 GB RAM.

An experimental study is carried out to ascertain the minimum average QPC (AQPC) for a data set comprising of 100 queries over 20 sites with a query length of 8 relations. Graphs were plotted showing the average QPC value of top-10, top-20, top-30 and top-40 query plans generated over 200 generations.

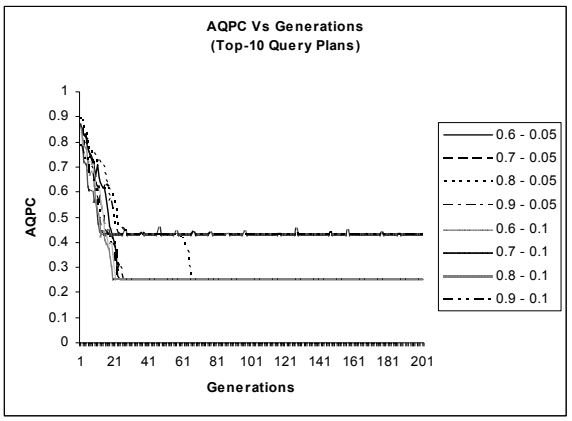
In each of the four graphs, a line chart is plotted for single point crossover with P_c in {0.6, 0.7, 0.8, 0.9} and mutation with P_m in {0.05, 0.1}. The graphs are shown in Figure 12. In all four graphs, the algorithm shows earlier convergence to the minimum AQPC value 0.25 for P_c=0.6, P_m=0.05 and P_c=0.8, P_m=0.1 .

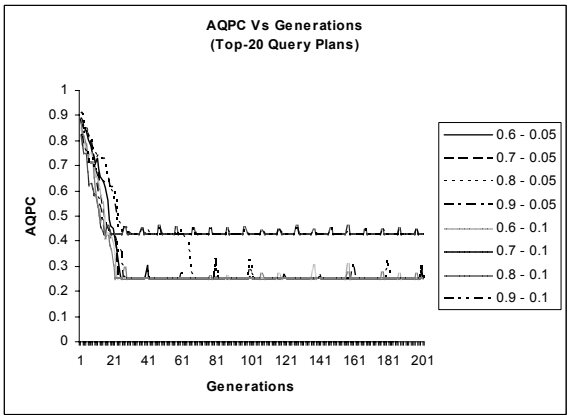
Further, in order to ascertain that P_c=0.6, P_m= 0.05 and P_c=0.8, P_m=0.1 are good for varied data sets, similar experiments were performed using these values for 5 different data sets. The corresponding graphs are shown in Figure 13 and Figure 14 respectively.

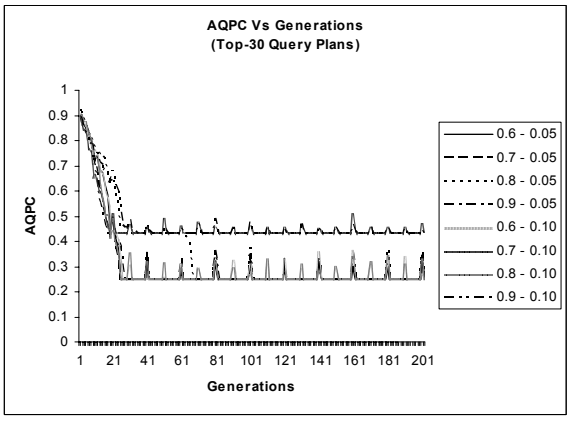
From the graphs in Figure 13 and 14, it can be inferred that the average QPC values of all 5 data sets converge to a minimum AQPC of 0.25 for P_c=0.6 and P_m= 0.05. Whereas for P_c=0.8 and P_m=0.1, three out of the five data sets converge to the minimum AQPC of 0.25. Therefore, Pc=0.6 and P_m= 0.05 are the most suitable values to generate top-k query plans with a minimum AQPC of 0.25.

Furthermore, graphs showing AQPC versus generations for top-10, top-20, top-30, top-40 and top-50 query plans were plotted. Four such graphs were plotted for 50, 100, 150 and 200 generations for P_c=0.6 and P_m= 0.05. These graphs are shown in Figure 15.

The graphs show that the top-k plans (k=10, 20, 30, 40, 50) converged to their optimal AQPC value of 0.25 in less than 50 generations whereafter no further improvement was observed for 100, 150 and 200 generations. This shows that the observed values of P_c=0.6 and P_m= 0.05 are the most fitting with respect to attaining top-k query plans with minimum AQPC in a lesser number of generations.







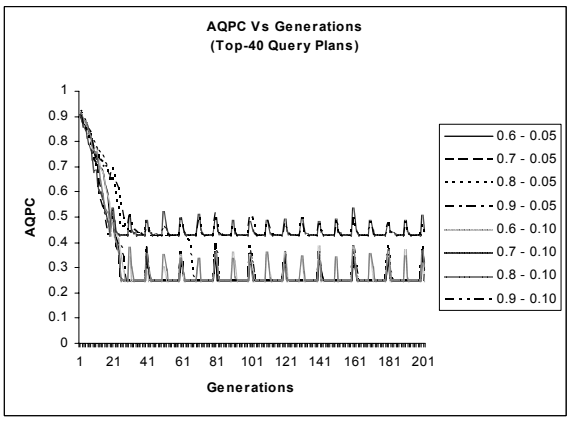
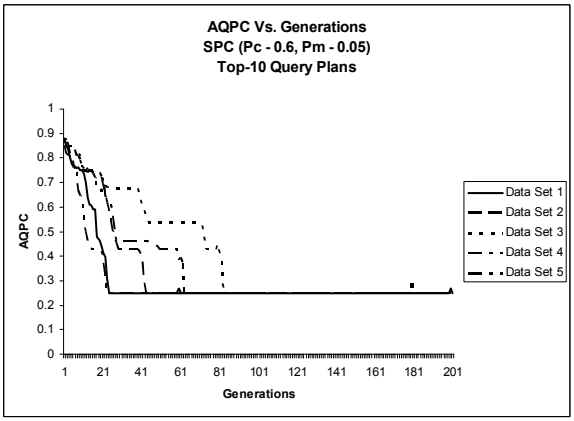
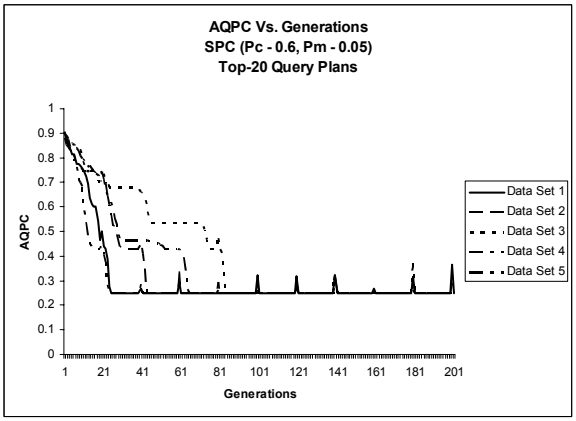
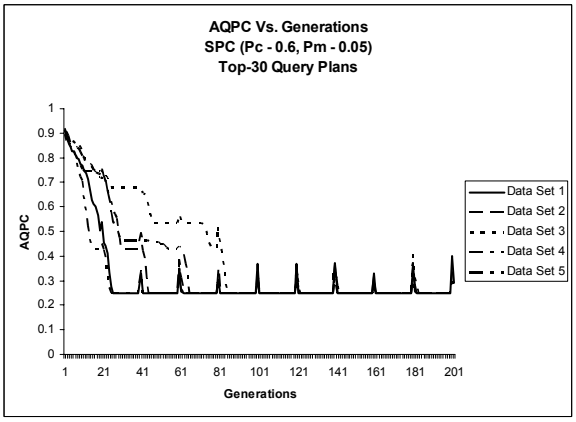


Figure 12. AQPC Vs. Generation – Single Point Crossover







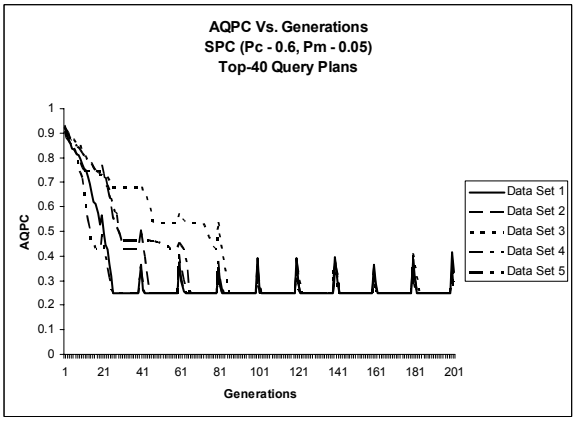


Figure 13. AQPC Vs. Generation – P_c=0.6 and P_m=0.05

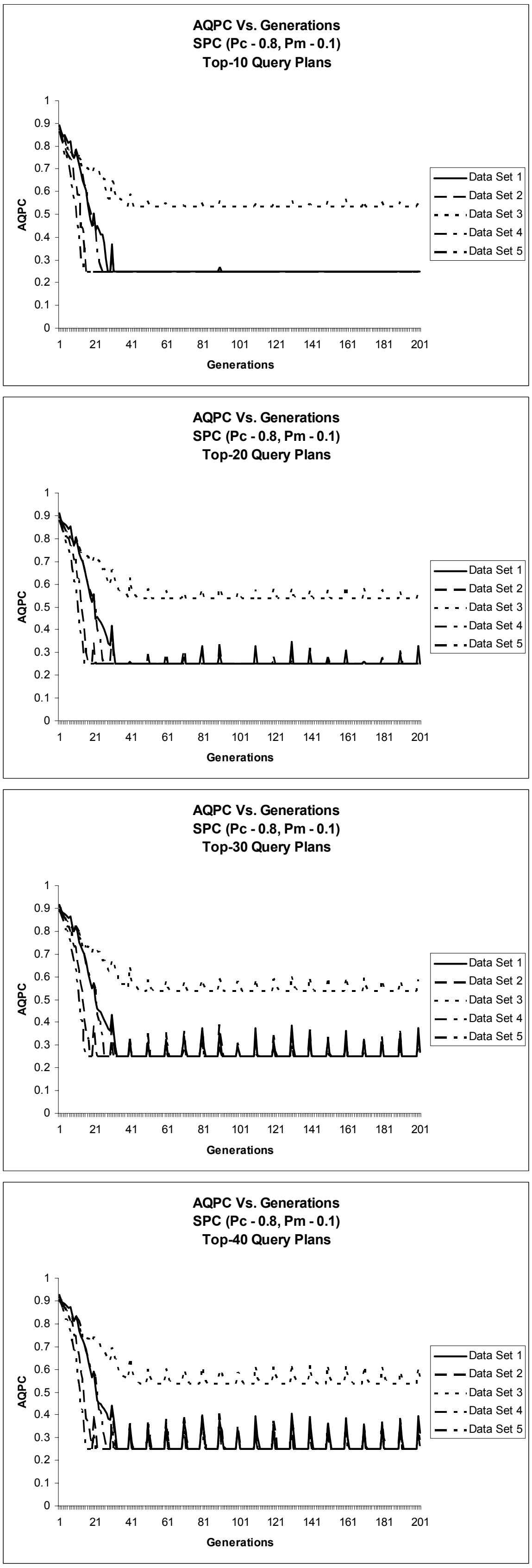


Figure 14. AQPC Vs. Generation – P_c=0.8 and P_m=0.1

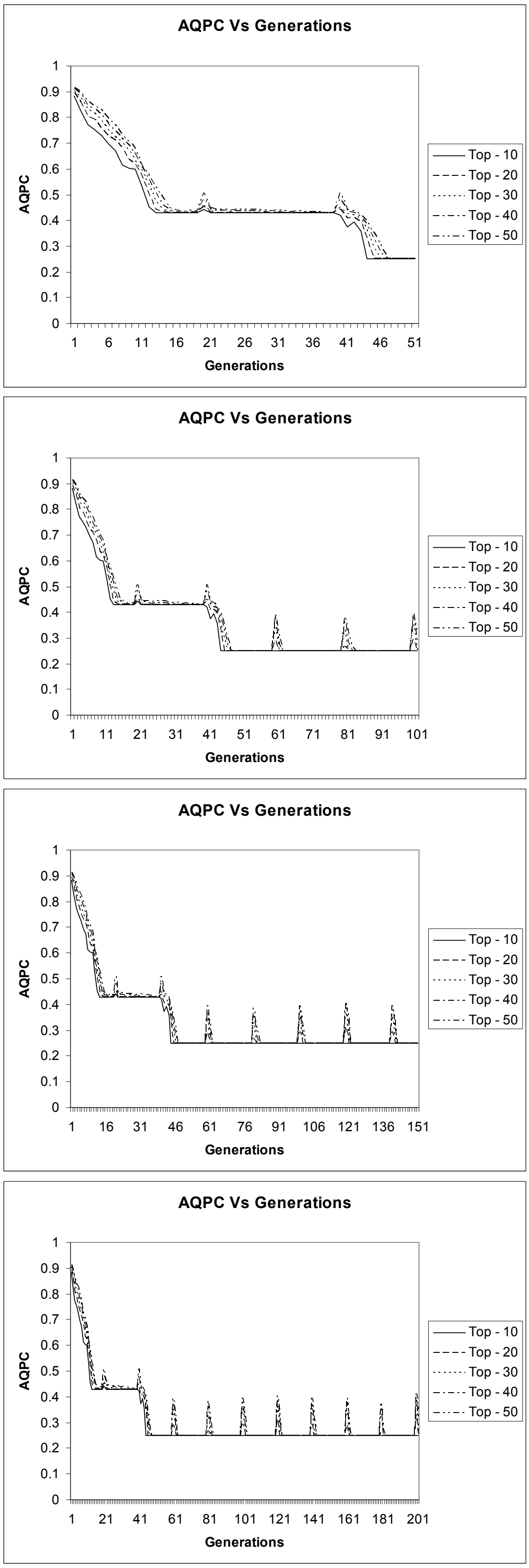


Figure 15. AQPC Vs. Generation – top-{10, 20, 30, 40, 50}

V. CONCLUSION

This paper presents an approach which attempts to generate distributed query processing plans that improve the response time of user queries. The approach achieves this by formulating the distributed query processing plan generation

as a single-objective genetic algorithm problem. The objective is to generate query plans with the required data, for answering the user queries residing close to each other i.e. query plans with minimum QPC. The query plans so generated involve fewer numbers of sites for answering the user query. As a result, query processing becomes efficient and, accordingly, facilitates decision making.

Further, experiments were carried out to determine the minimum average QPC values that can be achieved by the proposed approach. The results of these experiments show that the GA based algorithm converges quickly to minimum AQPC value for single point crossover with the observed crossover and mutation probabilities. These observations may be helpful in generating 'close' query plans for a given user query. The query plans so generated would result in efficient query processing.

REFERENCES

[1] K. Bennett, M.C. Ferris, and Y. Ioannidis, "A genetic algorithm for database query optimization, In proceedings of the Fourth International Conference on Genetic Algorithms, pp. 400-407, 1991

[2] P. Black, W. Luk, "A new heuristic for generating semi-join programs for distributed query processing," In Proceedings of the IEEE 6th International Computer Software and Applwation Conference (Chicago, Ill., Nov. 8-12), IEEE, New York, pp. 581-588, 1982

[3] P. Bodorik and J. S. Riordon, "Distributed Query Processing Optimization Objectives," Prec. of the IEEE Fourth Int. Data Engineering Conference, Los Angeles, CA, February 2-4, pp. 320-329, 1988

[4] P. Bodorik, J. S. Riordon, "A threshold mechanism for distributed query processing," Proceedings of the sixteenth annual conference on Computer science table of contents Atlanta, Georgia, United States, pp. 616 – 625, 1988.

[5] S. Ceri and G. Pelagati, "Distributed Database: Principles and Systems," McGraw Hill, 1984

[6] J. Chang, "A heuristic approach to distributed query processing," In Proceedings of the 8th Internatmnal Conference on Very Large Data Bases, VLDB Endowment, Saratoga, California, pp. 54-61, 1982

[7] K. Deb and S. Agrawal, "Understanding interactions among genetic algorithm parameters," in proceedings of the fifth workshop on foundation of genetic algorithms, Amsterdam, Netherlands, September 24-28, Morgan Kauffman, pp. 265–286, 1998

[8] H. Dong and Y. Liang, "Genetic algorithms for large join query optimization," In th proceedings of the Ninth annual conference on Genetic and Evolutionary Computation (GECCO), London, England, pp- 1211-1218, 2007

[9] M. Gregory, "Genetic algorithm optimization of distributed database queries", In Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, pp. 271-276, 1998

[10] A.R. Hevner and S.B. Yao, "Query processing in distributed database systems," IEEE Transactions on Software Engineering SE-5, pp. 177-187, 1979

[11] J. H. Holland, "Adaptation in natural and artificial systems," University of Michigan Press, 1975

[12] Y.E. Ioannidis and E. Wong, "Query optimization by simulated annealing,", in proceedings of the 1987 ACM-SIGMOD Conference, San Franscisco, CA, pp.9-22, 1987

[13] Y.E. Ioannidis and Y.C. Kang, "Left-deep vs. bushy trees: An analysis of strategy spaces and its implementations on query optimization," In SIGMOD International Conference on Managemeny of Data, Denver, pp. 168-177, 1991

[14] Y.E. Ioannidis and Y.C. Kang, "Randomized algorithms for optimizing large join queries, ACM 1990

[15] M. Jarke and J. Koch, "Query optimization in database systems," ACM Computing Surveys, volume 16, no. 2, pp. 111-152, June 1984

[16] Y. Kambayashi, M. Yoshikawa, and S. Yajima, "Query Processing for Distributed Databases using Generalized Semi-Joins, ACM, 1982

[17] Y. Kambayashi, M. Yoshikawa, Yagima, "Query processing for distributed databases using generalized semijoins," In Proceedings of the ACM-SIGMOD International Conference on Management of Data (Orlando, Fla., June 2-4) ACM, New York, pp. 151-160, 1982

[18] Y. Kambayashi, M. Yoshikawa, "Query processing utilizing dependencies and horizontal decomposition," In Proceedings of the ACM-SIGMOD International Conference on Management of Data (San Jose, Calif., May 23-26) ACM, New York, pp. 55-67, 1983

[19] D. Kosmann, "The State of the Art in Distributed Query Processing," ACM Computing Surveys, Vol. 32, No. 4, pp. 422-469, December 2000

[20] C. Liu, H. Chen, W. Krueger, "A Distributed Query Processing Strategy Using Placement Dependency," International Conference on Data Engineering, Vol. 0 (1996), 477, 1996

[21] C. Liu and C. Yu, "Performance issues in distributed query processing," IEEE transactions on parallel and distributed System, volume 4, No. 8, pp. 889-905, 1993

[22] M. Mitchell, "An Introduction to Genetic Algorithms," MIT Press, 1998

[23] M.T. Ozsu and P. Valduriez, "Principles of Distributed Database Systems", 2nd edition, Prentice Hall, Englewood Cliffs, N.J., 1991

[24] S. Rho and S.T. March, "Optimizing distributed join queries: A genetic algorithmic approach," Annals of Operations Research, 71, pp. 199-228, 1997

[25] A. Sokolov and D. Whitley, "Unbiased Tournament Selection," In proceedings of the 2005 conference on Genetic and Evolutionary Computation, pp. 1131-1138, 2005

[26] L. Stiphane, W. Eugene, "A State Transition Model for Distributed Query Processing,"ACM Transactions on database systems, Vol.11, No. 3, Pages 2, 1986

[27] A. Swami and A. Gupta, "Optimization of large join queries", In proceedings of 1988 ACM-SIGMOD Conference, Chicago, pp. 8-17, 1998

[28] M. Stillger and M. Spiliopoulou, "Genetic programming in database query optimization," In proceedings of the First Annual Conference on Genetic Programming, Stanford, CA, pp. 388-393, July 1996

[29] T.V. VijayKumar, Vikram Singh and Ajay Kumar Verma, "Generating Distributed Query Processing Plans using Genetic Algorithm", In the proceedings of the International Conference on Data Storage and Data Engineering (DSDE 2010), Bangalore, February 9-10, 2010, pp. 173-177, 2010

[30] C. T. Yu and C. C. Chang, "Distributed Query Processing," ACM Computing Surveys, volume 16, no. 4, pp. 399-433, 1984