Fragment Allocation using Genetic Algorithms in Distributed Database

Fatemeh Alavizadeh, Hamideh Sabouri

Abstract
Genetic algorithms are computing algorithms constructed in analogy with the process of evolution. The range of problems to which genetic algorithms have been applied is quite broad. Fragment allocation in distributed databases is a complex problem. The purpose of this work is solving this problem using genetic algorithms. We considered two different approaches. In the first method the solutions does not contain fragment duplication. But in the second one the fragments can be duplicated. The experimental results shows that the second approach is more time consuming but we can gain better results from it.

1. Introduction
The genetic algorithm is a method for solving optimization problems that is based on natural selection, the process that drives biological evolution. We can use genetic algorithms to solve a variety of optimization problems that are not well suited for standard optimization algorithms. One of these problems is fragment allocation in distributed databases.

Many different approaches are proposed for this purpose. In [1] the authors proposed a method which is designed to meet the requirements of clustering sites and determining fragment allocation in distributed database system, minimizing the communication cost between sites, and enhancing the performance in a heterogeneous network environment system. Clustering method is developed to group the sites into clusters, which helps in reducing the communication costs between the sites during allocation process. Fragment allocation method is developed to enhance system performance by increasing availability and reliability where multiple copies of the same fragments are allocated. In [2] the authors proposed a simple and comprehensive model that can reflect transaction behavior in distributed databases. Based on the model and transaction information, two heuristic algorithms have been developed to find a near-optimal allocation such that the total communication cost is minimized as much as possible. In [3] the authors introduced the genetic algorithm as a technique which has been used to obtain optimal and near optimal solutions to combinatorial problems. Son J. et al. [4] have introduced an adaptable vertical partitioning method in distributed systems. Yee W. et al. [5] proposed a method
of grouping fragments based on a measure of inter-client sharing, which describes how many fragments each client’s subscription has in common with those of others. Chun-Hung C. et al. [6] have explored the use of a genetic search-based clustering algorithm for data partitioning to achieve high database retrieval performance. They formulate the clustering problem in data partitioning as a Traveling Salesman Problem (TSP) and propose 2 genetic operators SE and SP, as well as modified version of ER operators to solve the associated (TSP). A vertical partitioning technique is used in their algorithm, and they show that their model is applied to solve the horizontal partitioning problem. Only one of the above approaches considered genetic algorithms for fragment allocation in distributed databases. In this method each fragment can be allocated to one site and we can not have duplication.

The purpose of this work is developing a genetic algorithm for fragment allocation in distributed databases with two different approaches. In the first approach we fragments can not be duplicated and in the second approach we can have fragment duplication. In the second approach we can gain better solutions but the number of possible solutions will become greater. In the next section there is a short introduction to genetic algorithms. In section 2 we explain our implementation. In section 4 we describe the results and in last section we conclude our work.

2. Genetic Algorithms
Genetic algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures to preserve critical information. An implementation of a genetic algorithm begins with a set of population of typically random chromosomes. One then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes, which represent a better solution to the target problem, are given more chance to reproduce than those chromosomes, which are poorer solutions. Genetic algorithms make it possible to explore a far greater range of potential solutions to a problem than do conventional programs.

3. Implementation
We considered three matrixes in our implementation. The requirement matrix stores integer values showing if a site requires a fragment or not. These values are between 0 and 100, indicating a weight for each requirement. The transmission cost matrix shows the cost for one site to access a fragment located in another site. The third matrix is update cost which is used in the second approach. This matrix indicates cost for a site to update a fragment in another site. Finally we considered an array for storing the capacity of each site. We considered a penalty for solutions which violate the site limits. As we describe earlier there are two different approaches for implementing a genetic algorithm in distributed databases for fragment allocation. Now we explain each of these methods separately in the following two subsections.

3.1. The First Approach
3.1.1. Planning a chromosome
At the heart of the genetic algorithm is the Chromosome. The Chromosome represents a potential solution and is divided into multiple genes. Step one; therefore, is to decide on the makeup of our chromosomes, which includes how many genes we want and what those genes will represent.
In the first method the length of each chromosome is equal to the number of fragments. Thus, each gene in the chromosome corresponds to a fragment. The value of a gene in the chromosome is an integer taking values between one and the number of sites and determines the number of the site which the related fragment is stored in. The structure of a chromosome is shown in figure 1.

![Figure 1. The structure of a chromosome in the first approach](image)

3.1.2. Fitness function
The role of the fitness function in a genetic algorithm is to determine the merit of each chromosome in a population. In the first method the cost of each chromosome is calculated with the following formula:

\[
\sum_{i=1}^{i=m} \sum_{j=1}^{j=m} \text{required}(i, j) \times \text{Transmission}(i, k)
\]

This formula calculates the cost of accessing all of the fragments for each site. In this formula i is the site number and j is the fragment number. Finally k is the number of the site which the fragment is stored in. Our attempt is to minimize this cost.

3.2. The Second Approach
3.2.1. Planning a chromosome
In the second method the length of each chromosome is equal to the number of fragments. Therefore every gene in chromosome corresponds to on fragment. The mean of the value of a gene is more complicated in this case. Actually the value of each gene is an n bit integer which n is the number of sites. Each bit in the gene is corresponds to one site. If the value of a bit is one, it means that the related fragment exists in the corresponding site. For example in the case of three sites, if the fragment is stored in the site 1 and site 3 the value of the related gene will be equal to 101 or 5. The structure of a chromosome is shown in figure 2.

![Figure 2. The structure of a chromosome in the second approach](image)
3.2.2. Fitness function
In the second method the cost of each chromosome is calculated using the following formula.

\[
\sum_{i=1}^{n} \sum_{j=1}^{m} \left( \min(\text{Transmission}(i,k)) \right) \text{required}(i, j) + \sum_{i=1}^{n} \text{update}(i,l) \text{required}(i, j)
\]

For calculating the cost of each chromosome we calculate the cost of accessing all of the fragments for each site. First we should calculate the transmission cost. For this purpose we should find all the sites which have a copy of the fragment. Then select the site which has the minimum transmission cost. After that we should calculate the update cost which is the sum of the update costs of all the sites which have a copy of this fragment. The attempt is minimizing this cost.

3.3. Implementation environment
We used the Java programming language in eclipse platform for the implementation. We used JGAP for genetic algorithm simulation. JGAP is a framework written in Java. JGAP is designed to do almost all of the evolutionary work in a relatively generic fashion. During the JGAP evolution process, chromosomes are exposed to multiple genetic operators that represent mating, mutation, etc. and then are chosen for the next generation during a natural selection phase based upon their fitness which is a measure of how optimal that solution is relative to other potential solutions. The entire goal of the genetic algorithm is to mimic the natural process of evolution in order to produce superior solutions. As mentioned earlier, a chromosome is made up of genes. JGAP lets us choose what gene class to use to represent each gene in the chromosome. That provides the most flexibility and convenience.

4. Results
We tested our genetic algorithms with different parameters. We considered different values for the initial population parameter and number of evolve and obtained the result. Initially we explain the results of the first approach. We began with a small problem in order to better visualize our results. In this problem there are 5 sites and 15 fragments. Figure 3 illustrates the network topology of this problem.

![Network Topology](image.png)

Figure 3. The network topology of the first problem
In this problem we considered a requirement matrix which its elements could take the value 1 or 0. We also considered that the transmission cost of each link in the network is equal to 1. Table 1 shows the result of finding a solution for this problem.

<table>
<thead>
<tr>
<th>Initial population</th>
<th>evolves</th>
<th>Sites limit</th>
<th>Minimum cost</th>
<th>Average cost</th>
<th>variance</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1000</td>
<td>3</td>
<td>26</td>
<td>30.5</td>
<td>7.83</td>
<td>2.79</td>
</tr>
<tr>
<td>100</td>
<td>10000</td>
<td>3</td>
<td>26</td>
<td>26.8</td>
<td>1.06</td>
<td>1.03</td>
</tr>
<tr>
<td>100</td>
<td>100000</td>
<td>3</td>
<td>23</td>
<td>24.6</td>
<td>1.15</td>
<td>1.07</td>
</tr>
<tr>
<td>100</td>
<td>1000</td>
<td>5</td>
<td>23</td>
<td>23</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table1.** The result of the first problem

The results show that increasing the number of evolves will lead to better average cost. Beside when we allocated more capacity to each site the genetic algorithm does a better functionality.

We repeat the test with another small problem. We have the same considerations as the previous problem for the matrixes. The network topology of this problem is shown in figure 4 and table 2 shows the results.

![Figure4. The network topology of the second problem](image)
Next we consider a more complex fragment allocation problem. In this problem we had 10 sites and 50 fragments. We considered a requirement matrix and a transmission cost matrix which their elements could take a value between 1 and 100. The result of this problem is shown in table 3. The result shows that increasing the initial population has a significant effect on getting better result.

<table>
<thead>
<tr>
<th>Initial population</th>
<th>evolues</th>
<th>Sites limit</th>
<th>Minimum cost</th>
<th>Average cost</th>
<th>variance</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1000</td>
<td>20</td>
<td>10234</td>
<td>10656.90</td>
<td>1780003.21</td>
<td>421.90</td>
</tr>
<tr>
<td>100</td>
<td>10000</td>
<td>20</td>
<td>9880</td>
<td>10522.50</td>
<td>143146.05</td>
<td>378.34</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>20</td>
<td>8282</td>
<td>8608.60</td>
<td>107224.8</td>
<td>327.45</td>
</tr>
<tr>
<td>500</td>
<td>10000</td>
<td>20</td>
<td>8116</td>
<td>8566.00</td>
<td>63812.44</td>
<td>252.61</td>
</tr>
</tbody>
</table>

Table3. The result of the third problem

Finally for the second approach we considered a problem with 5 sites and 15 fragments with a network topology similar to the first problem. First we increase the number of evolves with a fixed population size (figure 5). Then we increase the population size with a fixed number of evolves (figure 6). As it shown in the figures increasing the number of evolves and the number of initial population will lead to a lower cost. But after a threshold the cost does not change much. In this way we can gain the best values for these two parameters. At last we considered a suitable value for initial population and number of evolves and executed the genetic algorithm. The average cost of the algorithm for this case which the initial population size was 400 and the number of evolves was 10000, was equal to 159.90.
Figure 5. Effect of number of evolves on the cost

Figure 6. Effect of number initial population on the cost
5. Conclusion
In this paper we used genetic algorithms as an approach for fragment allocation in distributed databases. In this approach we made our algorithm more practical by adding the possibility of fragment duplication to the solutions. The result shows that changing the value of genetic algorithm input parameters will affect the final solution. Increasing the number of evolves and the number of initial population will lead to a lower cost. But after a threshold the cost does not change much. In this way we can gain the best values for theses two parameters.
In the future we can examine our algorithm with different crossover and mutation functions. Then we can compare the result with the current result and also choose the best functions for the purpose of fragment allocation in distributed databases.

Reference: