A GENETIC ALGORITHM FOR FRAGMENT ALLOCATION IN A DISTRIBUTED DATABASE SYSTEM*

Arthur L. Corcoran John Hale

The University of Tulsa

Keywords: *Genetic Algorithms, Distributed Databases.*

Abstract

In this paper we explore the distributed database allocation problem, which is intractable. We also discuss genetic algorithms and how they have been used successfully to solve combinatorial problems. Our experimental results show the GA to be far superior to the greedy heuristic in obtaining optimal and near optimal fragment placements for the allocation problem with various data sets.

1 Introduction

Computerized databases have become an essential part of our lives. They play a critical role in nearly all areas where computers are used. A few of the areas include business, engineering, science, medicine, law, and education. Traditionally, databases and database management systems (DBMS) have resided on a single site. This is called a centralized database system. Recently, there has been a rapid trend toward distributed models of computation, where several remote sites are connected via a communications network. Distributed database systems (DDBS) and distributed database management systems (DDBMS) have been developed in response to this trend. For convenience, we will use the term *distributed databases* (DDBs) to refer to DDBSs and DDBMSs, collectively. The advantages of distributed databases include greater reliability and availability as well as improved performance. Unfortunately, distributed databases are accompanied by increased overhead and complexity in the system design and implementation. This complexity is often combinatorial in nature.

Genetic algorithms (GAs) provide an excellent technique for dealing with the combinatorial problems found in distributed databases. GAs borrow the techniques and mechanisms from genetics and natural evolution to effectively find optimal and near-optimal solutions to complex and difficult problems.

*Research partially supported by OCAST Grant AR2-002, OCAST Grant AR2-004, NSF Grant IRI-9110709 and Sun Microsystems, Inc.

This paper is organized as follows: Section 2 provides an introduction to distributed databases. This includes a discussion of the advantages and disadvantages of DDBs. A simple DDB model is presented which proves to be intractable. Section 3 provides an introduction to genetic algorithms. Section 4 describes our application of a GA to the DDB problem as well as the experimental results obtained. Finally, Section 5 provides a summary and conclusions.

2 Distributed Databases

Distributed databases is the term we use collectively for distributed database systems and distributed database management systems. These systems were developed in response to the current trend toward distributed computing. Unlike traditional centralized database systems, DDBs are spread over many sites. These sites are connected by a communications network.

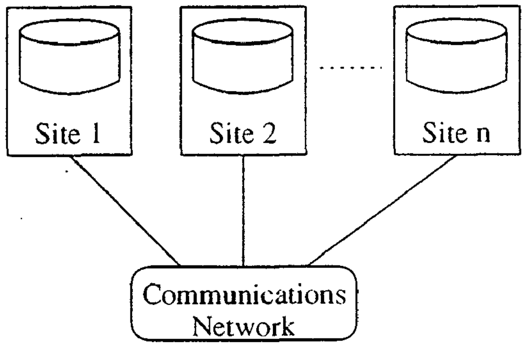

Figure 1: A Typical Distributed Database

Figure 1 illustrates the architecture of a typical distributed database. Portions of the entire database are spread out over multiple computers, called *sites* or *nodes*. The computers are connected by a communications network with a given topology. Each local site may have its own local database, which can be maintained by a traditional DBMS. Each site may also contain *fragments*, or portions of the distributed global database. Fragments are managed by application and communication processing software.

Some of the advantages of DDBs include reliability and availability. Reliability is loosely defined as the probability that a system is up at a particular moment in time. Availability refers to the probability that a system is continuously available during some time interval. In a traditional centralized database system, the failure of the single site means failure of the entire system. In a DDB, the failure of a single site will only effect access to data located at that site. Clearly, this leads to improved reliability and availability. Another advantage of DDBs is the performance improvement obtained by distributed processing. Local queries and transactions accessing data at a single site are much faster since the local database is smaller. Transactions involving different sites can be processed concurrently, reducing execution and response time. This is especially an advantage when the database is naturally distributed over different locations, such as in a business with databases used by regional offices which are all accessible from the corporate headquarters. These types of database systems are typically dominated by local queries and transactions. Finally, DDBs allow sharing of data while at the same time retaining localized control. This can be an important issue in database security when maintaining a 'need to know' authorization scheme.

A potential drawback in a DDB is the added complexity and overhead involved in its design and implementation. The DDB must be designed to preserve consistency in the database yet provide acceptable response time for transactions across many different sites. Strategies must be developed to handle distributed queries and transactions. The distribution design step involves fragmentation of relations and allocation of these fragments. The objective of fragmentation is to achieve better units of distribution. Allocation is concerned with optimal placement of the fragments among the available sites. Special care must be taken in the placement of replicated fragments to maintain consistency and access efficiency. Finally, the DDB must be able to gracefully recover from failures such as site crashes or network hangups.

The additional functionality and flexibility in a DDB is a difficult problem to deal with. Finding optimal solutions is a step beyond. In addition to the normal database design issues and the fragmentation process, the designer of a distributed database must also decide on how to distribute the fragments over the sites. We now present a formal description of a simple distributed database allocation problem.

A distributed database is composed of a collection S of m sites, where each site i is characterized by its capacity, c_i,

$$S = \{c_1, c_2, c_3, ..., c_i, ..., c_m\},$$

and a set F of n fragments, where each fragment j is characterized by its size, s_j,

$$F = \{s_1, s_2, s_3, ..., s_j, ..., s_n\}.$$

Each fragment is required by at least one of the sites. The site requirements for each fragment are indicated by the requirements matrix,

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,n} \end{bmatrix}$$

where $r_{i,j}$ indicates the requirement by site i for fragment j. In general, this requirement is represented by a real value, that is, a weight. A variation of this is to use a boolean value to indicate that fragment j is either required or not required by site i. Transmission cost is given by the transmission cost matrix,

$$T = \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,m} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ t_{m,1} & t_{m,2} & \cdots & t_{m,m} \end{bmatrix}$$

where $t_{i,j}$ indicates the cost for site i to access a fragment located on site j.

Given the above definitions, the distributed database allocation problem is one of finding the optimal placement of the fragments at the sites. That is, we wish to find the placement,

$$P = \{p_1, p_2, p_3, ..., p_j, ..., p_n\}$$

(where $p_j = i$ indicates fragment j is located at site i) for the n fragments so that the capacity of any site is not exceeded,

$$\sum_{j=1}^{n} r_{i,j} s_j \leq c_i \qquad \forall i | 1 \leq i \leq m$$

and the total transmission cost,

$$\sum_{i=1}^{m} \sum_{j=1}^{n} r_{i,j} t_{i,p_j}$$

is minimized.

By restricting the use of the requirements matrix and having zero transmission cost, the distributed database allocation problem can be transformed to the bin packing problem, which is known to be NP-complete [7]. The DDB allocation problem is considerably more difficult than bin packing, so it is clearly also NP-complete. Consequently, unless an efficient algorithm has been found to solve intractable problems and it is proven that P = NP, then we must turn to heuristic methods to obtain approximate solutions. Also, in this paper we ignore exhaustive methods such as branch and bound due to their inability to solve large combinatorial problems.

For a more detailed treatment of databases and distributed databases, the reader is referred to works by Bell [1], Bell and Grimson [2], Ceri *et al.* [3], Chang and Shielke [4], Elmasri and Navathe [6], and Özsu and Valduriez [10].

3 Genetic Algorithms

A *genetic algorithm* (GA) is an adaptive search technique based on the principles and mechanisms of natural selection and 'survival of the fittest' from natural evolution. GAs grew out of Holland's [9] study of adaptation in artificial and natural systems. By simulating natural evolution, in this way, a GA can effectively search the problem domain and easily solve complex problems. Furthermore, by emulating biological selection and reproduction techniques, a GA can perform the search in a general, representation-independent manner.

The genetic algorithm operates as an iterative procedure on a fixed size population or pool of candidate solutions. The candidate solutions represent an encoding of the problem into a form that is analogous to the chromosomes of biological systems. Each chromosome represents a possible solution for a given objective function. Associated with each chromosome is a fitness value, which is found by evaluating the chromosome with the objective function. It is the fitness of a chromosome which determines its ability to survive and produce offspring. Each chromosome is made up of a string of genes (whose values are called alleles). The chromosome is typically represented in the GA as a string of bits. However, integers and floating point numbers can easily be used.

The GA begins by generating an initial population, $P(t = 0)$, and evaluating each of its members with the objective function. While the termination condition is not satisfied, a portion of the population is selected, somehow altered, evaluated, and placed back into the population. At each step in the iteration, chromosomes are probabilistically selected from the population for reproduction according to the principle of the 'survival of the fittest'. Offspring are generated through a process called crossover, which can be augmented by mutation. The offspring are then placed back in the pool, perhaps replacing other members of the pool. This process can be modeled using either a 'generational' [8, 9] or a 'steady-state' [12] genetic algorithm. The generational GA saves offspring in a temporary location until the end of a generation. At that time the offspring replace the entire current population. Conversely, the steady-state GA immediately places offspring back into the current population.

4 Experimental Results

We developed a genetic algorithm for the distributed database problem using LibGA [5]. The problem was encoded so that each gene in the chromosome corresponds to a fragment. An integer representation was used in which the allele values correspond to site locations. For example, an allele value of 5 in gene 7 would indicate the placement of fragment 7 at site 5. This corresponds to the placement vector, P, in Section 2. Initial allele values were selected at random, ranging from 1 to the number of sites, m. It is possible with this encoding scheme to have infeasible solutions, that is, solutions which violate the site capacity constraints or which place fragments in inaccessible sites. Consequently, our objective function calculated the proper cost for feasible solutions and used a penalty for the infeasible solutions. The penalty depended on the number of sites, m. For each fragment placement which violated a constraint, the objective function added a penalty of $500m$ to the fitness. Such a mild penalty balances the preservation of beneficial genetic material with the selection pressure of feasibility [11].

The genetic algorithm was tested with several different parameters. We used two different population models: generational and steady-state. We also used three different crossover operators: simple (one point), uniform, and asexual. Note, asexual crossover is simply a swap of two randomly selected genes. When mutation occurred, a randomly selected gene was replaced with a randomly selected choice from the range of valid allele values. The mutation rate was fixed at 0.1, and the pool size was fixed at 500. Note, all of our reported results represent feasible solutions.

We began with a small problem in order to better visualize our results. A problem was generated which had 5 sites and 15 fragments to allocate. The fragment size was fixed at 1 and the site capacity was fixed at 3. Consequently, the final result has no wasted site capacity and exactly 3 fragments per site. A requirements matrix was generated with each fragment required by a randomly selected site. Additional requirements were generated randomly with a 21% probability that a particular fragment is required at any particular site. Network topology was also generated randomly with a 70% probability that any two sites are adjacent. Transmission cost was fixed at 1 unit per hop.

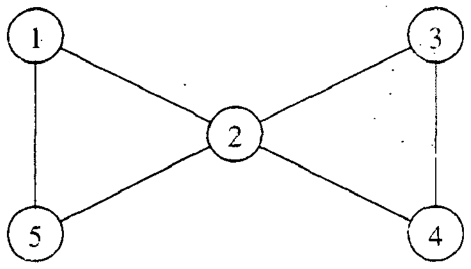

Figure 2: 'Bow Tie' Network Topology

Figure 2 illustrates the resulting network topology generated. Coincidentally, this topology resembles a 'bow tie'. The requirements were as follows:

Site	Required Fragments
1	6, 9, 10, 12, 13, 14
2	7, 11
3	3, 4, 5, 6, 10, 12, 13, 14
4	2, 4, 5, 8, 9, 10, 11, 14
5	1, 2, 3, 6, 10, 15

We applied a greedy heuristic to this problem which places each fragment in turn in the least cost location. The greedy heuristic determined the following allocation:

Site	Fragments
1	6, 9, 10
2	2, 3, 7
3	4, 5, 12
4	8, 11, 13
5	1, 14, 15

with a total transmission cost of 27. This allocation places 4 fragments at sites in which they are not required.

Model	Crossover	x_{min}	$\bar{x}$	σ^2	σ
Gen.	Simple	26	30.10	4.322	2.079
	Uniform	24	26.70	2.678	1.636
	Asexual	23	23.00	0	0
SS.	Simple	25	27.40	1.822	1.350
	Uniform	24	25.80	3.511	1.874
	Asexual	23	23.00	0	0

Table 1: Results for 'bow tie' data set

Table 1 summarizes the results we obtained with the GA. For each reproduction model and crossover operator, the table lists the best result obtained (x_{min}) after running the

GA with 10 different random seeds. Other columns list the average ($\bar{x}$), variance (σ^2), and standard deviation (σ) of the 10 runs. From the x_{min} values, we see the GA easily outperforms the greedy heuristic. However, on the average, simple crossover did worse than greedy under both models. Uniform crossover performed slightly better on average than greedy. Asexual crossover was the best performing crossover, consistently obtaining apparently optimal results. The following is an allocation generated by the GA:

Site	Fragments
1	9, 12, 13
2	7, 10, 11
3	3, 5, 14
4	2, 4, 8
5	1, 6, 15

This allocation places 2 fragments at sites in which they are not required, half as many as by the greedy.

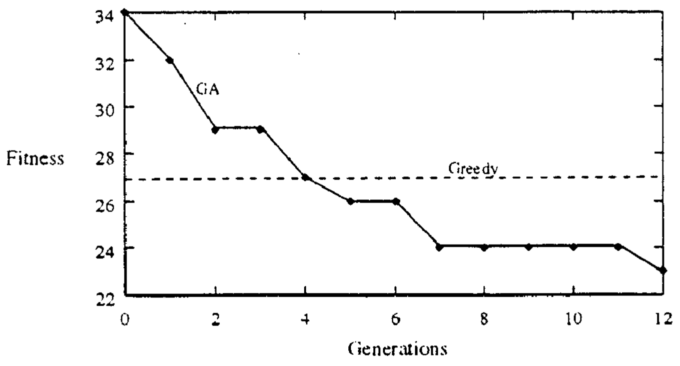

Figure 3: Convergence Profile (Bow Tie)

Figure 3 illustrates the convergence profile of the genetic algorithm. The greedy result is indicated by the dashed line. While the GA begins with a worse result than greedy, it is able to quickly converge to a better answer.

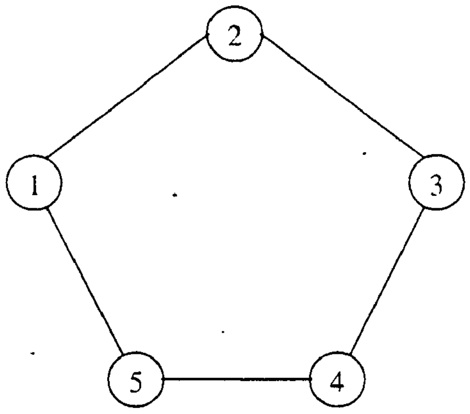

Figure 4: 'Ring' Network Topology

Figure 4 illustrates the next problem we examined. In this case we changed the network topology to a ring and left the other parameters identical to the 'bow tie' problem. For this problem, the greedy heuristic resulted in the following placement of the fragments:

Site	Fragments
1	6, 9, 12
2	7, 11, 13
3	3, 4, 5
4	2, 8, 10
5	1, 14, 15

with a total transmission cost of 24. In this case the greedy only placed two fragments at sites in which they were not required.

Model	Crossover	x_{min}	$\bar{x}$	σ^2	σ
Gen.	Simple	28	30.40	2.044	1.430
	Uniform	24	26.80	3.956	1.989
	Asexual	23	23.00	0	0
SS.	Simple	24	27.70	4.011	2.003
	Uniform	25	26.00	0.8889	0.9428
	Asexual	23	23.10	0.1000	0.3162

Table 2: Results for 'ring' data set

Table 2 lists the results for the GA on the 'ring' data set. Under this topology, the simple and uniform crossovers could only equal the performance of the greedy at best, and were much worse on average. Asexual crossover proved to be the best, with apparently optimal results under both models. The following is an assignment generated by the GA:

Site	Fragments
1	6, 9, 13
2	7, 11, 12
3	3, 5, 14
4	2, 4, 8
5	1, 10, 15

The GA placed only one fragment at a site in which it was not required. This was half as many as in the greedy.

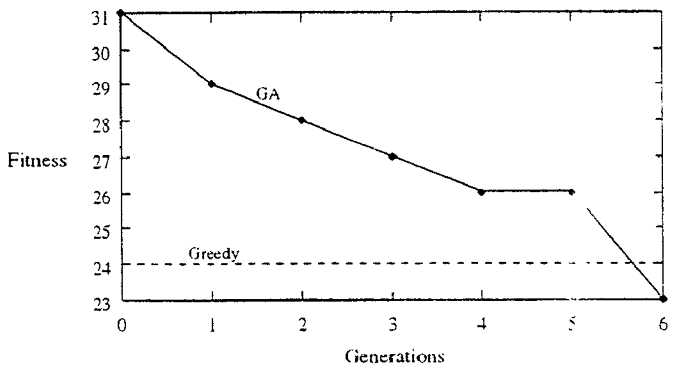

Figure 5: Convergence Profile (Ring)

Figure 5 illustrates the convergence profile for the GA on the ring data set. As before, the GA started with a worse answer than the greedy but quickly converged to the optimal. However, for this data set the greedy was able to obtain nearly optimal results.

We next turned our attention to a complex fragment allocation problem. We generated a data set with 20 sites and

50 fragments. This corresponds to a search space of 20^{50}, or about 10^{65} possible solutions. Not all of these solutions are feasible. Fragment sizes were randomly generated in the range from 1 to 10. Site capacities were randomly generated in the range from 20 to 40. The probability that more than one site required a fragment was 40% and the probability that any any two sites were adjacent was 30%. Transmission cost was randomly generated in the range from 1 to 10. All of these values and ranges were chosen arbitrarily. For this data set, the greedy obtained a placement with total transmission cost of 2014.

Model	Crossover	x_{min}	$\bar{x}$	σ^2	σ
Gen.	Simple	1978	2007.40	237.4	15.41
	Uniform	1972	1985.70	113.8	10.67
	Asexual	1952	1958.70	31.79	5.638
SS.	Simple	2036	2065.80	821.3	28.66
	Uniform	2001	2030.00	394.9	19.87
	Asexual	1990	2017.80	506.6	22.51

Table 3: Results for 20 site data set

Table 3 summarizes the results obtained with the GA on the 20 site data set. The best GA result was better than the greedy in all cases except for the steady-state model with simple crossover. The greedy was able to beat the average performance of the steady-state model under all crossover operators. However, under the generational model, the average performance of all crossover operators was able to beat the greedy. The clear victor for this data set was asexual crossover under the generational model.

As a final test of our GA, we ran it on a variation of the 20 site data set. This new data set was generated using the same parameters as before except that there were 100 fragments and site capacities were generated in the range from 50 to 55. With 100 fragments and 20 sites, this corresponds to a search space of 20^{100}, or about 10^{130} possible solutions, not all of which are feasible. The greedy heuristic obtained 4142 for this data set. We only tested the generational GA with asexual crossover on this data set as it is clearly the best choice. We obtained the following results over ten runs:

x_{min}	$\bar{x}$	σ^2	σ
4013	4027.80	81.51	9.028

As before, the GA was a clear winner over the greedy heuristic. With this data set we see the GA's solution quality did not degrade as the search space size was increased.

5 Conclusions

In this paper, we have explored the distributed database allocation problem, which is intractable. We introduced the genetic algorithm as a technique which has been used to obtain optimal and near optimal solutions to combinatorial problems. We found the GA to have superior performance to the greedy heuristic on fragment allocation problems of various sizes. While the greedy heuristic took time and effort to implement, the GA was very straightforward: an encoding was decided upon, and a simple function was written to evaluate candidate solutions. We found the best parameters for the GA to be the use of a generational reproduction model with asexual crossover. This is most likely due to the fact that the fitness landscape is rather rugged, and since asexual crossover is much like a mutation, it performs well on such rugged landscapes.

In future, we plan to extend our DDB allocation problem to include explicit replication of fragments. We have found that the GA allows us to easily obtain solutions to the allocation problem and is easily extended to the solution of other related problems.

Acknowledgements

This research has been partially supported by OCAST Grant AR2-002, OCAST Grant AR2-004, and NSF Grant IRI-9110709. The authors also wish to acknowledge the support of Sun Microsystems, Inc.

References

[1] D. A. Bell. Difficult data placement problems. *Computer J.*, 27(4), 1992.

[2] D. A. Bell and J. Grimson. *Distributed Database Systems*. Addison-Wesley, Menlo Park, California, 1992.

[3] S. Ceri, G. Martella, and G. Pelagatti. Optimal file allocation for a distributed databasee on a network of minicomputers. In *Proc. ICOD 1 Conf.*, Aberdeen, Scotland, 1980.

[4] C. C. Chang and Shielke. On the complexity of the file allocation problem. In *Conf. on Foundations of Data Organisation*, Kyoto, Japan, 1985.

[5] A. L. Corcoran and R. L. Wainwright. LibGA: A user-friendly workbench for order-based genetic algorithm research. In E. Deaton, K. M. George, H. Berghel, and G. Hedrick, editors, *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 111–118, New York, 1993. ACM Press.

[6] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, Redwood City, California, 1989.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[9] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan, 1975.

[10] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[11] J. T. Richardson, M. R. Palmer, G. E. Liepens, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, Arlington, Virginia, 1989. Morgan Kaufmann.

[12] D. Whitley and J. Kauth. GENITOR: A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, Denver, Colorado, 1988.