Database Design with Genetic Algorithms

Walter Cedeño
Independent Consultant
446 Creekside Drive
Downingtown, PA 19335
wcedeno@voicenet.com

V. Rao Vemuri
Department of Applied Science
University of California, Davis &
Lawrence Livermore National Lab.
Livermore, CA 94550
vemuri1@llnl.gov

Abstract

The design of distributed databases requires a configuration of the data such that queries are satisfied by accessing a minimum number of locations and the system load is equitably distributed among all locations. This problem is called the File Design Problem. This problem is NP Hard and requires the optimization over conflicting objectives. A genetic algorithm based on *multi-niche crowding* combines heuristics with parallel processing to provide a suitable approach to solve this problem. Performance of the algorithm is tested using multiple data sets on different system platforms. The new method holds promise in providing suitable solutions for this problem.

Keywords: Database design, multimodal functions, genetic algorithms, heuristics, parallel processing, optimization

1. Introduction

Consider a distributed database system with multiple nodes, as shown in Figure 1, that contains all the data for a company-wide database comprising millions of records. In order to use the resources equitably and efficiently, the data in the database must be organized so that queries for database records are balanced among all nodes. That is, all nodes must handle about the same number of queries on average. Additionally, the information in the database must be organized so that a query can be satisfied by accessing only a small number of nodes. Minimizing the number of nodes accessed on a single query reduces greatly the communication overhead on the network. This requirement asserts that records with the same attributes should be placed together, if possible, at one location. These two criteria must be balanced to obtain an optimal database configuration. This problem is known as the File Design Problem (FDP).

The File Design Problem is known to be NP-hard. The goal is to find an assignment of database records to files that minimizes the average number of files examined over all single attribute queries. Techniques using Artificial Neural Networks (Liang *et. al.,* 1991) have been applied to this problem in the past. In this work we describe a solution to the File Design Problem using a Genetic Algorithm (GA, Holland, 1975). In particular we describe the application of the Multi-Niche Crowding Genetic Algorithm (MNC GA; Cedeño, 1995) to this problem. Our implementation of the MNC GA is written

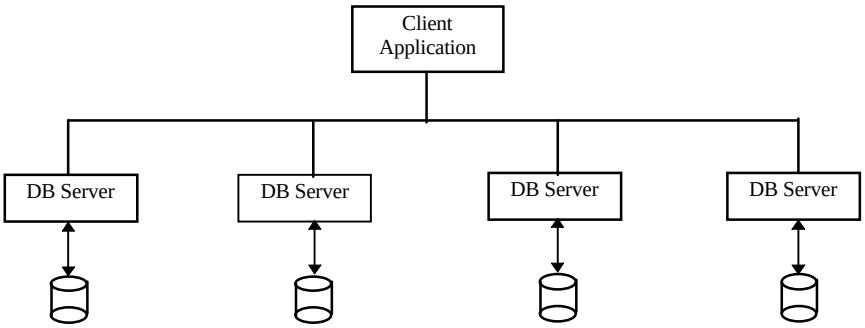

Figure 1: Example of a distributed database system

in SISAL (Streams and Iterations in a Single Assignment Language; McGraw, 1985), a functional language that takes advantage of parallel architectures. Using the portability and architecture independence inherent in SISAL a parallel model of the MNC GA is defined that provides increased performance without losing the convergence properties of the algorithm.

Additionally we introduce the use of heuristics in the crossover and mutation operators used during the mating step. These operators prove to be essential in finding optimal solutions to the FDP. The use of heuristics and the ability of the MNC GA to search for multiple conflicting database configurations provides us a promising hybrid approach for solving complex combinatorial problems. Results with various test cases are shown. Performance of the algorithm is shown for different computer platforms.

This paper is organized as follows. Section 2 presents an overview of the MNC GA model used to solve this problem. Section 3 describes the File Design Problem in detail and presents examples. Section 4 describes the heuristic based genetic operators used for this problem. Section 5 defines the experimental setup used to test the performance of the MNC GA. Section 6 describes the results and the performance of the approach. Finally, Section 7 contains some comments and discussion about the applicability of the method.

2. The Multi-Niche Crowding GA

DNA, the building block of every living creature provides organisms a way to evolve and adapt to changing environments. Only organisms well adapted to their environment can survive from one generation to the next, transferring on the traits, that made them successful, to their offspring. Competition for resources and the ever changing environment drives some species to extinction and at the same time others evolve to maintain the delicate balance in nature.

The ability of organisms to evolve and adapt to their environment by means of natural selection has provided mother nature with a diverse set of species. This foundation, which is part of modern evolutionary thinking, was laid by Charles

Darwin after the publication of his work “On the Origin of Species by Means of Natural Selection”. Only organisms well adapted to their environment can survive from one generation to the next, transferring on the traits that made them successful to their offspring. Competition for resources between organisms and the ever changing environment drives some species to extinction and at the same time others evolve to maintain the delicate balance in nature. It is through this interaction between nature and organisms, that species containing favorable traits for a given environment emerge. In this work we apply the same principles present in nature to create a genetic algorithm that evolves a population of mathematical solutions containing different categories of solutions adapted to niches in a multimodal environment.

In this Section we describe the MNC GA, a computational metaphor to the survival of species in ecological niches in the face of competition. The MNC GA maintains stable subpopulations of solutions in multiple niches in multimodal landscapes. The algorithm introduces the concept of *crowding selection* to promote mating among members with similar traits while allowing many members of the population to participate in mating. The algorithm uses *worst among most similar replacement* (WAMS) policy to promote competition among members with similar traits while allowing competition among members of different niches as well.

The benefits of an approach that can locate multiple optima and maintain them throughout the search are many. Consider, for example, a dynamic environment where the optima are constantly changing, a technique that can locate and maintain multiple optima can inform the user when the current configuration is no longer the best based on the parameters in the environment. More details can be found in Cedeño (1995), Cobb & Grefenstette (1993), Dasgupta & McGregor (1992), Goldberg & Smith (1987), and Ng & Wong (1995). In other cases abnormal situations may cause changes in the current configuration; having viable alternatives at hand can allow for a more smoother transition to the new configuration. An approach that can use a set of solutions to locate multiple optima is more practical for these types of environments. Additionally, there exist many problems where the location of the best K optima are needed in order to compare different answers and point out further experimentation. The benefits of the MNC GA have been already shown in applications to problems in DNA mapping (Cedeño, Vemuri, and Slezak, 1995) and aquifer management (Cedeño and Vemuri, 1996)

Figure 2 shows an overview of the MNC GA. Initially, all the individuals in the population (size n) are created at random and evaluated in parallel. Once the initial population is created, the operations of selection, mating and mutation, and replacement are applied for a given number of generations. In each generation all individuals in the population are selected for mating and their mates are chosen in parallel using crowding selection. Then in parallel, all n pairs participate in mating producing $2n$ offspring. The $2n$ offspring undergo mutation and those different than their parent are allowed to participate in replacement. The offspring left are then inserted, one at a time, into the

population using WAMS replacement. These steps are repeated for the specified number of generations.

```
Generate initial population of size n at random.
Evaluate initial population.
For gen = 1 to MAX_GENERATIONS
  Use crowding selection to find mate for all individuals
  Mate and mutate all pairs
  Insert offspring in population using WAMS replacement
```

Figure 2: Overview of parallel application of the Multi-Niche Crowding GA.

In the MNC GA both the selection and replacement steps are modified with some type of crowding (De Jong, 1975). The idea is to eliminate the selection pressure caused by fitness proportionate reproduction (FPR) and allow the population to maintain diversity throughout the search. This objective is achieved in part by encouraging mating and replacement within members of the same niche while allowing some competition for the population slots among the niches. The result is an algorithm that (a) maintains stable subpopulations within different niches, (b) maintains diversity throughout the search, and (c) converges to different local optima. No prior knowledge of the search space is needed and no restrictions are imposed during selection and replacement thus allowing exploration of other areas of the search space while converging to the best solutions in the different niches.

2.1 Crowding Selection

In MNC, the FPR selection is replaced by what we call *crowding selection*. In the parallel application of crowding selection used in this work each individual in the population gets a chance for mating in every generation. Application of this selection rule is done in two steps. First, each individual I_i from the population is selected as a parent for mating. Second, its mate I_j is selected, not from the entire population, but from a group of individuals of size C_s (crowding selection group size), picked uniformly at random (with replacement) from the population. The mate I_j thus chosen must be the one who is the most "similar" to I_i. The similarity metric used here is not a genotypic metric such as the Hamming distance, but a suitably defined phenotypic distance metric.

Crowding selection promotes mating among members having similar traits and allows all the members of the population to participate in mating. This allows members of the same niche to participate in mating more often and preserve those traits that define their species. At the same time mating between different species may occur giving rise to new species. Unlike mating restriction (Deb and Goldberg, 1989) that only allows individuals from the same niche to mate, crowding selection allows some amount of exploration to occur while at the same time looking for the best individual in each niche.

2.2 Worst Among Most Similar Replacement

During the replacement step, MNC uses a replacement policy called *worst among most similar* (WAMS). The goal of this step is to pick an individual from the population for replacement by an offspring. Implementation of this policy follows these steps. First, C_f "crowding factor groups" are created by picking uniformly at random (with replacement) s (crowding group size) individuals per group from the population. Second, one individual from each group that is most similar to the offspring is identified. This gives C_f individuals that are candidates for replacement by virtue of their similarity to the offspring. The offspring will replace one of them. From this group of most similar candidates, we pick the one with the lowest fitness to die and be replaced by the offspring. Figure 3 shows a pictorial view of this replacement policy.

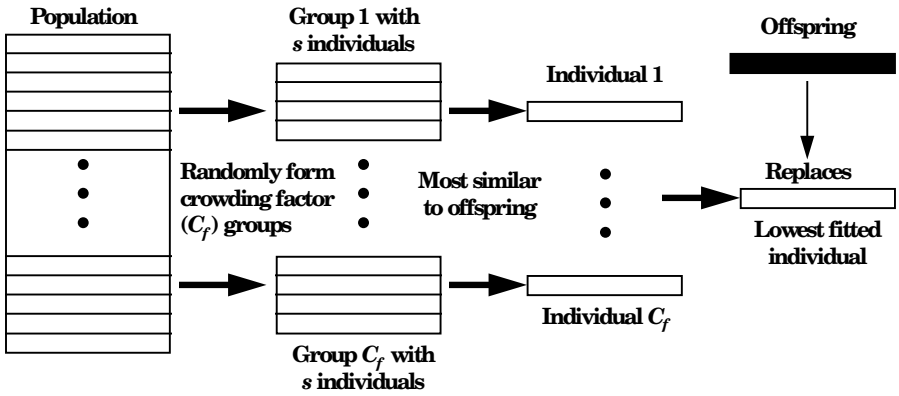

Figure 3: Worst among most similar (WAMS) replacement policy

After the offspring becomes part of the population it competes for survival with other individuals when the next offspring is inserted in the population. In WAMS replacement offspring are likely to replace low fitness individuals from the same niche. It can also happen that it replaces a high fitness individual from the same niche or an individual from another niche. This allows a more diverse population to exist throughout the search. At the same time it promotes competition between members of the same niche and between members belonging to different niches. A similar technique was used by Goldberg (1989) in classifier systems but he replaced the most similar individual out of a group of low fitness candidates.

Worst among most similar replacement promotes competition among members with similar traits belonging to the same niche while allowing competition among members of different niches as well. This replacement technique accomplishes two things. First, by promoting competition among members of the same species in a niche it applies the *survival of the fittest* rule that is so prevalent in nature. Only those that are fit to their environment can survive for many generations, thus allowing the species to evolve to their best potential within their niche. Second, by allowing competition between different species as

well, those species that are a better fit for their environment tend to occupy more slots in the overall population.

Both the selection and replacement steps in the MNC are primarily based on a similarity metric. Fitness is also considered during replacement to promote competition among members of the same niche. Competition among members of different niches occurs as well.

3. The File Design Problem

The File Design Problem is an NP-hard problem; that is, the number of possible solutions increases exponentially as the problem size increases linearly. It arises in the context of database design for a distributed system.

The problem is defined as follows. Given a set of *N* records, and each is characterized by a single attribute **A** that takes *h* different values { a_1, a_2, ..., a_h}. There are n_i records corresponding to attribute a_i, i.e., $n_1 + n_2 + ... + n_h = N$. The assumption is made that queries for records of any given attribute are equally likely. The query distribution is used only during the calculation of the fitness function, as we will show in Section 4.4. Other, more practical, distributions can be applied easily by modifying the fitness function accordingly, without affecting the behavior of the MNC GA. We also have *K* files of size *b* such that $K * b = N$. Different file sizes can also be accomodated to represent realistic configurations. The approach presented here is not limited by the simplifications made to the problem. The constants *K*, *b*, n_i, *N*, and *h* are all positive integers.

The problem is to find an assignment of the *N* records to the *K* files such that the average number of files (*ANF*) accessed over all possible single-attribute queries is minimized. In other words, an assignment of the records to the files must be found such that (on average) queries for the records with the same attribute can be satisfied by reading from as few files as possible.

Table 1: Possible configurations for 12 records in 2 files of size 6. The attributes values are {B, C, F, V} and have {2, 7, 1, 2} records respectively.

File 1	File 2	fex(B)	fex(C)	fex(F)	fex(V)	ANF
C C C C C C	C B B V V F	1	2	1	1	1.25
C C C B V V	C C C C B F	2	2	1	1	1.50
C C C C B V	C C C B V F	2	2	1	2	1.75
C C C B B F	C C C C V V	1	2	1	1	1.25

For example, consider a database with $N = 12$ employee records characterized by their last name (here the last name refers to attribute **A**). Moreover, assume that all records have a last name in the set **A** = {Blattner, Cedeño, Feo, Vemuri} with n = {2, 7, 1, 2} records respectively. Here we have a total of $h = 4$ possible last names (attribute values). These records will be placed in a database consisting of $K = 2$ files of size $b = 6$. The problem now is to save the employee

records in the database such that queries for records with a given last name (attribute value) access the minimum number of files (on average). Using the first letter of each last name the first two columns in Table 1 shows some sample configurations for this example.

The *ANF* for a configuration is given by the formula

$$\sum_{i=1}^{h} fex(a_i) \Big/ h,$$

where the function called $fex(a_i)$ returns the number of files that must be accessed to retrieve all the records with attribute a_i. From Table 1, the second configuration has a value of 2 for $fex(B)$ since both files contain a record with attribute value B. The first and last configurations in the table are examples of optimal solutions for this problem. Even though the *ANF* values are the same, in some contexts the last solution is better because it has a more balanced configuration. If requests for the attributes are distributed uniformly, file 1 and file 2 will be accessed 25% and 100% of the time respectively in the first solution, where as the last solution will be accessed 50% and 75% of the time respectively. This idea is incorporated when evaluating solutions generated by the MNC GA.

GAs have been successfully applied to a variety of optimization problems, such as the Traveling Salesman Problem (Whitley *et. al.*, 1989), Scheduling (Syswerda and Palmucci, 1991; Michalewicz, 1992), and the Bin Packing Problem (Falkenauer and Delchambre, 1992). In some cases better results were obtained when the mating operator was designed to capture the essential information in the problem. With this in mind, the mating operator for the File Design Problem was designed using the “first fit” and “best fit” heuristics (to be described later). Such heuristics, group records with the same attribute together. The multimodal search space in the problem is explored in many directions by using selection and replacement operators in the MNC GA that encourages mating and replacement between solutions from the same extrema.

In this work we apply a parallel version of the generational MNC GA. The intent is two fold. First, we want to show that the generational version of the MNC GA exhibits the same properties as its steady state counterpart. Second, we want to show the advantages of the parallel version of the generational MNC GA, namely, the straight forward implementation on parallel architectures. SISAL was selected as the language for the parallel implementation because it is portable and easy to learn. The application can be ported to multiple platforms, including SGIs, Crays, and SUNs. Performance can then be evaluated using different number of processors. Additionally, SISAL is a deterministic functional language which guarantees the same solutions on different platforms.

There are basically three parallel GA models (Gordon et. al., 1992) exhibiting different degrees of parallelism; fine grain, distributed, and direct. In a fine grain model (Davidor, 1991; Gorges-Schleuter, 1989; and Spiessens & Manderick, 1991), each solution in the population is mapped to a processor with

genetic operators applied between nearest neighbors. In a distributed model (Mühlenbein et. al., 1991; Tanese, 1989), processors are assigned subpopulations, which converge locally and exchange genetic material among them at fixed intervals. Direct models (Grefenstette, 1981), exploit the parallelism inherent in the GA operators and the GA structure while having the same properties of a sequential GA. Our SISAL implementation of the MNC GA follows the direct model while having the localized convergence exhibited in the other models.

The parallelism inherent in the generational MNC GA, and in the operators, is easily exploited. Performance is enhanced while maintaining the necessary computation for solving the problem. The best solution was found in all test cases with a speedup of at least 2.2 with four processsors.

4. The Genetic Algorithm Model for the File Design Problem

The SISAL version of the MNC GA was designed to capture the parallelism in the model while maintaining the search for multiple solutions. In this model, multimodality is exploited by encouraging mating and replacement between solutions from the same peak. Improved performance is obtained by creating the offspring in parallel. The offspring are then inserted into the population sequentially to preserve replacement between members of the same peak.

The solutions in the initial population are created in parallel by assigning records to files at random. There are K files with b slots each for a total of N slots. The slots are uniquely numbered with a value between 1 and N. Each record is then assigned a slot number corresponding to a unique position in a file. The constraints of the problem are easily maintained without the need for counters for each of the files. The *fitness*, a measure of "goodness" of a solution, is then calculated for each member of the population.

The algorithm is executed for a fixed number of generations. Each generation consists of creating all the offspring and inserting them into the population. Three steps are involved to create two offspring: select the parents, apply the mating operator to the parents, and calculate the fitness to the offspring. Mutation is applied by the mating operator as part of the mating process. Each offspring is inserted sequentially in the population by selecting an existing solution to die.

To create the offspring each solution in the population is selected as a parent. This allows every individual in the population to mate at least once in every generation. All the mates for the parents are selected in parallel using crowding selection. After selection, mating produces two offspring and their fitness are computed. The number of offspring created can be up to two times the number of solutions in the population. We create a total of n (population size) mating pairs and each pair produces 2 offspring with probability χ (crossover probability). All offspring are created in parallel with a given crossover and mutation probability.

The offspring are inserted one at a time in the population using the worst among most similar (WAMS) replacement policy. Replacement is applied sequentially. After an offspring is inserted in the population it immediately becomes a candidate for replacement and must compete with the other solutions in the population to survive. Some offspring are indeed replaced in the same generation before getting a chance to reproduce. As in selection, the replacement operator is biased toward solutions within the same extrema. Convergence is improved by the replacement operator which eliminates solutions with lower fitness.

The following sections describe the encoding and genetic operators for the File Design Problem. They were designed to preserve the constraints of the problem and take full advantage of the implementation of SISAL arrays.

4.1 Chromosome Encoding

A *chromosome* represents a valid solution to the problem. It consists of an array of *N alleles* corresponding to each of the records in the problem. Each allele may assume a value between 0 and *K* - 1 inclusive, indicating the file containing the record. A valid encoding is a *N* digit number in base *K* where all digits appear exactly *b* times. An example is shown in Figure 4 for the records defined in Table 1. To make clear that the chromosomes are not binary we selected in this case *K* = 3 files of size *b* = 4.

Record number:	1	2	3	4	5	6	7	8	9	10	11	12
Record attribute:	B	B	C	C	C	C	C	C	C	F	V	V
Chromosome 1:	0	0	0	0	1	1	1	1	2	2	2	2
Chromosome 2:	0	0	1	1	2	2	2	0	0	1	1	2

Figure 4: Encoding for the file design problem.

4.2 Similarity Metric

Similarity between two solutions is measured from the number of records assigned to the same file. An example is shown in Figure 5 using the data from Table 1. As in the previous section we have 3 files of size 4. Each digit indicates the file where a record is located. In this example we have a similarity value of 8, that is 8 records have been assigned to the same file.

Records:	B	B	C	C	C	C	C	C	F	V	V
Chromosome 1:	1	**0**	2	**2**	**0**	**1**	**0**	**1**	1	**0**	2	**2**
Chromosome 2:	2	**0**	1	**2**	**0**	**1**	**0**	**1**	2	**0**	1	**2**
Similar assignments:		**1**		**2**	**3**	**4**	**5**	**6**		**7**		**8**

Figure 5: Using similarity to select a mate during crowding selection.

4.3 Mating Operator

The crossover operator for the File Design Problem creates two offspring and was designed with two goals in mind. First, the characteristics expressed in both

parents will be expressed in the offspring, thus preserving the schemata in both solutions. Second, fitness should be improved when combining two similar solutions. “Best fit” and “first fit” heuristics (described later) are used for this. Incorporating these features in the mating operator improves convergence of solutions from the same extrema. When two solutions from different extrema mate, offspring from other extrema can be created. This way the operator is not restricted to small areas in the search space.

The first step in the mating operator is to transfer similar characteristics from the parents to the offspring. This is done by transferring the records assigned to the same file in both parents to the same file in the offspring. Those records not assigned are counted for each attribute and sorted in decreasing order. One offspring is created using a best fit method based on the contents of files. In this approach unassigned records are located into files where records with the same attribute reside. The main idea is to group files with the same attribute in the same file as much as possible. The second offspring is created using a first fit method based on the empty space in the files. Here the unassigned records will be located where file space is available for records with the same attribute. Using the configuration in Table 1 an example is shown in Figure 6.

```
Offspring inherits similar alleles from parents:
  Record Attribute: B B C C C C C C C F V V
          Parent 1: 0 0 1 2 2 2 0 1 0 1 2 1
          Parent 2: 0 1 0 1 2 2 1 2 0 0 1 2
         Offspring: 0 - - - 2 2 - - 0 - - -
Unassigned records by attribute: B:1, C:4, F:1, V:2


Assignment of records in sorted order to both
offspring:
           Offspring 1                Offspring 2
         Best Fit Method            First Fit Method
C:4  0 - 2 2 2 2 0 0 0 - - -   0 - 1 1 2 2 1 1 0 - - -
V:2  0 - 2 2 2 2 0 0 0 - 1 1   0 - 1 1 2 2 1 1 0 - 0 0
B:1  0 1 2 2 2 2 0 0 0 - 1 1   0 2 1 1 2 2 1 1 0 - 0 0
F:1  0 1 2 2 2 2 0 0 0 1 1 1   0 2 1 1 2 2 1 1 0 2 0 0
```

Figure 6: Mating operator for the File Design Problem

Given the parents in Figure 6, the offspring inherits only four alleles; 3 records with attribute **C** and 1 record with attribute **B**. Using the best fit method the other 4 records with attribute **C** are assigned to file 2 and file 0 because those files contain records with the same attribute. Using the first fit method the 4 records are assigned to file 1 because that file is the most empty and all records can be placed together. If not all records fit in one file then the remaining records are placed in the next file having the most available space. The next

attribute having the highest number of unassigned records is selected and its records are assigned in a similar manner.

Mutation is applied with a fixed probability for each allele. When an allele is selected for mutation another position in the chromosome is selected at random and the two values are interchanged. Such mutations may introduce a new configuration in succeeding generations.

4.4 Fitness Function

The fitness function captures three important characteristics of an optimal solution: low *ANF*, records with the same attribute are grouped together, and records with the same attribute are spread equally among the minimum number of files needed to store them. The last two points are captured in a grouping term (*GT*) and balancing term (*BT*) respectively. The two terms are contradictory in the sense that *GT* wants to group records together, while *BT* wants to spread records equally across files. These three terms are added together, with different weight values, to get the fitness of a solution. The *GT* value is given a higher weight over the *BT* value because it promotes lower *ANF* values in the solution.

Recall from Section 3 that the *ANF* value is given by the formula:

$$ANF = \sum_{i=1}^{h} fex(a_i) \Big/ h ,$$

where $fex(a_i)$ returns the number of files containing attribute a_i. From this formula we can compute an upper and lower bound to the *ANF* term. The lower bound represents a configuration where the records for all attributes are assigned to the least number of files needed to contain them. The upper bound can be calculated from the configuration containing the records for all attributes spread among the maximum number of files possible. The lower and upper bound are called *min_anf* and *max_anf* respectively and are:

$$\min\_anf = \sum_{i=1}^{h} \lceil n_i / b \rceil \Big/ h \le ANF \le \max\_anf = \sum_{i=1}^{h} \min(n_i, K) \Big/ h .$$

Recall that n_i denotes the number of records with attribute i, *h* denotes the number of attributes, *b* denotes the size of the files, and *K* denotes the number of files.

To compute the *GT* value we need to know how the records of a given attribute are spread in the files. Since we want as many records as possible of the same attribute grouped together, we came up with an equation that looks at the ratio of records with the same attribute in each file. The *GT* value is computed by adding the normalized number of records squared for each attribute in every

file. The higher the number of records of the same attribute in a file the higher the *GT* value. The formula for the *GT* value is:

$$GT = \sum_{i=1}^{h} \sum_{j=1}^{K} \left(attr(a_i, j) / n_i \right)^2 ,$$

where $attr(a_i,j)$ returns the number of records of attribute a_i in file j.

On the other hand the *BT* value wants to spread the records with the same attributes equally among the minimum number of files needed to fit the records. The *BT* value is then computed by adding the absolute value of the difference between the number of records for each attribute and a balance configuration for the attribute. Only files containing records for the given attribute are included in the summation. The formula for this term is:

$$BT = \sum_{i=1}^{h} \sum_{j=1}^{K} \left\lfloor \left| attr(a_i, j) - n_i / \lceil n_i / b \rceil \right| \right\rfloor, \text{ when } attr(a_i, j) \neq 0.$$

Here $\lceil n_i / b \rceil$ returns the number of files needed to store the records of attribute a_i. Values of *BT* closest to zero represent more balanced configurations.

The three terms *ANF*, *BT*, and *GT* are used to define the fitness value for a solution. Since higher positive values are used to indicate a better solution, the terms are normalized to return values between 0.0 and 1.0. A percentage of each term is then added to form the final fitness value as indicated by the following formula:

$$fitness = 0.70 * \frac{GT}{h} + 0.25 * \frac{\max\_anf - ANF}{\max\_anf - \min\_anf} + 0.05 * \frac{1.0}{1.0 + BT}.$$

The fitness value for any solution is a number between 0.0 and 1.0. Solutions where the fitness value is 1.0 represent configurations where the *min_anf* value is achievable and all the records for any attribute can fit in the minimum number of files. Having the property of fitting records with the same attribute in one file eliminates the conflict between *BT* and *GT* while obtaining a maximum value of *h* for *GT*.

5. Experimental Data

To evaluate the behavior of the algorithm six test cases, having different properties, were created. Some of the test cases contain solutions achieving the *min_anf* lower bound. In other test cases we have attributes with the number of records exceeding the file size (therefore the fitness < 1.0 and a *min_anf* configuration may not exist). In all test cases, multiple attributes per file were

mixed to create the different configurations. For all configurations 100 records were used. Table 2 summarizes all configurations created.

Table 2: Configuration for all test cases

Case Num.	Num. Files	File Size	Num. Attr.	Number of Records per Attribute n_1 n_2 n_3 n_4 n_5 ... n_h	*min_anf* exist
1	5	20	10	7, 2, 3, 1, 5, 17, 18, 13, 15, 19	Yes
2	10	10	10	7, 2, 3, 1, 5, 17, 18, 13, 15, 19	Yes
3	5	20	10	7, 4, 3, 8, 6, 11, 18, 15, 10, 18	No
4	10	10	10	7, 4, 3, 8, 6, 11, 18, 15, 10, 18	No
5	5	20	21	7, 4, 3, 8, 6, 1, 8, 5, 10, 8, 1, 2, 4, 9, 5, 1, 6, 2, 3, 3, 4	Yes
6	5	20	15	7, 4, 9, 7, 7, 4, 9, 5, 9, 7, 9, 5, 6, 7, 5	No

To evaluate the performance of the implementation three different platforms were used: the SGI Iris 4D, Cray Y-MP, and Cray C90. The execution time from one to four processors was collected for the algorithm using case 1 in Table 2. The MNC GA parameters used for each run are:

Population size:	100
Number of generations:	50
Mating probability:	0.95
Mutation probability:	0.01
C_s for selection:	4
C_f for replacement:	3
s for replacement:	5

These parameters were chosen after a trial and error period. They represent a good set of choices for the test data shown in Table 2.

6. Results

The generational MNC GA was very successful for the test data in Table 2. For all test cases, multiple optimal solutions were found and retained for many generations. In four of the six test cases at least one optimal solution was found prior to generation 6. More generations were needed for the test cases 3 and 4. These test cases have the property that the *min_anf* is not achievable and there are attribute values where the number of records is higher than the file size. In those cases, the solutions were competing between themselves for a very small improvement in fitness.

Table 3 shows solutions for all test cases and the generation number on which they were obtained. Each solution is represented by the file number to which each record is assign. The records with the same attribute value are separated by

commas to verify how many files are used to save them. For example, the first solution has the records for attribute 1 assigned to file 4, records with attribute 2 assigned to file 1, and so on.

Table 3: Best solution and the number of generations needed for all test cases

Case	Gen	Best Solution
1	3	4444444, 11, 333, 0, 22222, 33333333333333333, 111111111111111111, 4444444444444, 222222222222222, 0000000000000000000
2	4	9999999, 77, 111, 6, 88888, 33333333113311111, 222222222277777777, 5555555555999, 444444444488888, 0000000000666666666
3	25	2222222, 0000, 222, 44444444, 333333, 44444444444, 000000000100001000, 333333433333333, 2222222222, 111111111111111111
4	15	8888888, 9999, 888, 55555555, 999999, 22222222227, 000000000066666666, 111111111155667, 3333333333, 444444444477777777
5	5	0000000, 3333, 222, 33333333, 222222, 0, 00000000, 11111, 2222222222, 44444444, 2, 44, 4444, 11111111, 3333, 4, 111111, 44, 333, 444, 0000
6	5	0000000, 2222, 222222222, 4444444, 222222, 3333, 333333333, 44444, 000040000, 3333333, 111111111, 11111, 111111, 4444444, 00000

By examining test case 4 more closely we can observe that the optimal configuration required the records for the eighth attribute (111111111155667) assigned to 4 different files. All other attribute values were assigned to 1 or 2 files only. In the same run other configurations were found were the *ANF* was the same and all the records for the eight attribute were stored in 2 or 3 files. In such cases other attribute values were assign to 3 or 4 files.

In general, the use of heuristics improved the convergence of the MNC GA. We tried other crossover operators, but they required many more generations to achieve similar results. At the same time, the MNC GA did not allow the population to converge prematurely to a local optimum. Mixing heuristics with the GA allowed us to obtain results which are better than using the heuristics alone. Heuristics alone tend to locate local optima frequently, whereas the MNC GA allows different solutions to converge at the same time giving a higher likelihood to obtain the global optimum, as defined by the fitness function, in search spaces with multiple optima.

Platform	1 Proc.	2 Proc.	3 Proc.	4 Proc.
Y-MP C90	12.9799	8.6748	6.4425	5.4930
Y-MP	18.6106	10.4642	8.9153	6.3648
SGI Iris	25.8900	16.5300	13.4200	11.9000

Figure 7: MNC GA execution time in seconds for 50 generations.

We also observed the increase in speed that can be obtained using a parallel implementation of the MNC GA. A speedup between 2.2 and 2.9 was achieved with four processors in the three different platforms. Figure 7 summarizes the performance from one to four processors in the different platforms.

The best speedup was obtained for the Cray Y-MP platform and the worst speedup on the SGI Iris platform. The speed of the Cray is not much faster than that of the SGI when we take into account that the SGI does not have vector calculations and has worse cache locality. Better speedup times can be obtained for more complex (in terms of evaluation time) fitness functions. This is because selection and mating are done in parallel whereas replacement is done sequentially. Since the fitness of a new offspring is calculated at the end of the mating step a more complex fitness function will benefit from the parallelism.

7. Summary

The results obtained with the parallel version of the generational MNC GA model are encouraging. Diversity was maintained during the run, just as the steady state algorithm did, and the last generation contained multiple optima. Exploiting the multimodality inherent in the File Design Problem resulted in a more balanced search over the entire space.

Creating genetic operators that use heuristics enhanced the convergence of the algorithm while at the same time allowing multiple solutions to coexist. In this work we developed a model from the problem's point of view. We enhanced the MNC GA with problem specific operators to provide a better way to search for optimal configurations. The convergence to optimal solutions was achieve in all cases while improving the performance using SISAL.

A better speedup can be achieved using more complex fitness functions or by introducing a parallel version of the *WAMS* replacement operator. The parallel version must retain the important properties of WAMS. Competition among solutions within the same peak is encouraged while allowing competition among the multiple peaks as well.

Like the steady state MNC GA, the algorithm located and maintained multiple solutions throughout the run while maintaining diversity in the population. Creating genetic operators that use heuristics enhanced the convergence of the algorithm while at the same time allowing multiple solutions to coexist. These operators enhanced the ability of the MNC GA to search for optimal configurations. The convergence to optimal solutions was achieve in all cases while improving the speed with a parallel version developed using SISAL. In the future we want to investigate in more detail the use of heuristics for genetic operators.

References

Cedeño, W. (1995). The multi-niche crowding genetic algorithm: analysis and applications. *UMI Dissertation Services*, 9617947.

Cedeño, W., Vemuri, V., and Slezak, T. (1995). Multi-Niche crowding in genetic algorithms and its application to the assembly of DNA restriction-fragments. *Evolutionary Computation*, 2:4, 321-345.

Cedeño, W. and Vemuri, V. (1996). Genetic algorithms in aquifer management. *Journal of Network and Computer Applications*, 19, 171-187, Academic Press.

Cobb, H. J. and Grefenstette, J. J. (1993). Genetic algorithms for tracking changing environments. In S. Forrest (ed.) *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers San Mateo, California, 523-530.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection*.

Dasgupta, D. & McGregor, D. R. (1992). Non-stationary function optimization using the structured genetic algorithm. In R. Manner and B. Manderick (eds.), *Parallel Problem Solving from Nature*, 2. Amsterdam: North Holland, 145-154.

Davidor, Y. (1991). A naturally occurring niche & species phenomenon: The model and first results. In R. K. Belew and L. B. Booker, (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 257-263.

De Jong, K. A. (1975). An analysis of the behaviour of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan. *Dissertation Abstracts International* 36(0), 5140B. (University Microfilms No. 76-9381).

Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization, In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, 42-50.

Falkenauer, E. and Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*.

Gorges-Schleuter, M. (1989). ASPARAGOS an asynchronous parallel optimization strategy. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 422-427.

Goldberg D. E. & Smith R. E. (1987). Non-stationary function optimization using genetic algorithms with dominance and diploidy. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms.* Hillsdale, NJ: Lawrence Erlbaum Associates

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Reading MA: Addison-Wesley.

Gordon, V.S., Whitley, D., and Böhm, A.P.W. (1992). Dataflow parallelism in genetic algorithms. In R. Manner and B. Manderick (eds.), *Parallel Problem Solving from Nature 2*, Elsevier Science Publishers.

Grefenstette, J.J. (1981). Parallel adaptive algorithms for function optimization. Technical Report No. CS-81-19, Vanderbilt University, Computer Science Department.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*, Ann Arbor MI: The University of Michigan Press.

Liang, J., Chang, C. C., Lee, R. C. T., and Wang, J. S. (1991). Solving the file design problem with neural networks. *Tenth Annual International Phoenix Conference on Computers and Communications*.

McGraw, J., et. al. (1985). SISAL - Streams and iterations in a single-assignment language. Language reference manual, version 1.2. Lawrence Livermore National Laboratory manual M-146 (Rev. 1), Livermore, CA.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. New York, NY: Springer-Verlag.

Mühlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimization. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 416-421.

Ng, K. P. & Wong, K. C. (1995). A new diploid scheme and dominance change mechanism for non-stationary function optimization. In L. J. Eshelman (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*. San Mateo, CA:Morgan Kaufmann, 159-166.

Spiessens, P. and Manderick, B. (1991). A massively parallel genetic algorithm - implementation and first analysis. In R. K. Belew and L. B. Booker, (Eds.), *Proceedings Fourth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 279-287.

Syswerda, G. and Palmucci, J. (1991). The application of genetic algorithms to resource scheduling. In R. K. Belew and L. B. Booker, (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, 502-508.

Tanese, R. (1989). Distributed genetic algorithms, In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 434-440.

Whitley, D., Starkweather, T., & Fugway, D. (1989). Scheduling problems and traveling salesmen: the genetic edge recombination operator. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, 133-140.