A Massively Distributed Parallel Genetic Algorithm (mdpGA)

Shumeet Baluja

13 October 1992

CMU-CS-92-196R

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Abstract

The effectiveness of combinatorial search heuristics, such as Genetic Algorithms (GA), is limited by their ability to balance the need for a diverse set of sampling points with the desire to quickly focus search upon potential solutions. One of the methods often used to address this problem is to simulate the theory of punctuated equilibria in the GA. The GA introduced here uses the basic premises derived from punctuated equilibria, but hopes to remedy the problems associated with sudden introduction of new genetic material by relying upon a much greater degree of distribution and an overlapping population architecture. Presented here is a description and preliminary empirical test results of a massively distributed parallel genetic algorithm (mdpGA) . On the test problems attempted, the mdpGA did significantly better than a simple parallel GA in terms of speed and the number of evaluations required to find the optimal solutions. The massive distribution of the GA and the modified population topology prove to be far less vulnerable than other genetic algorithms to biases in the function space which lead away from global optima.

This research was partly sponsored by Defense Advanced Research Projects Agency, under contracts “Perception for Outdoor Navigation” (contract number DACA76-89-C-0014, monitored by the US Army Topographic Engineering Center) and “Unmanned Ground Vehicle System” (contract number DAAE07-90-C-R059, monitored by TACOM). It was also partially sponsored by the National Science Foundation, under NSF Contract BCS-9120655, titled “Annotated Maps for Autonomous Underwater Vehicles”, and the NSF grant titled “Massively Parallel Real-Time Computer Vision”. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the National Science Foundation, or the U.S. Government.

Keywords: Genetic Algorithms, Punctuated Equilibria, Massive Parallelism, DeJong's Function Suite.

1. Introduction

1.1. Genetic Algorithms

Genetic algorithms (GA) are general purpose optimization tools designed to search irregular, poorly characterized function spaces. The GA is established upon the foundations of natural selection and genetic recombination. A GA combines the principles of survival of the fittest with a randomized information exchange. Although the information exchange is randomized, the GA is far different than a simple random walk. A GA has the ability to recognize trends toward optimal solutions, and exploit such information by guiding the search toward them.

A genetic algorithm maintains a population of potential solutions to the objective function being optimized. In much of the GA research done, the potential solutions have been encoded as binary bit strings. The initial group of potential solutions is determined randomly. These potential solutions, termed "chromosomes", are allowed to evolve over a number of generations. At every generation, the fitness of each potential solution is calculated. The fitness is a measure of how well the potential solution optimizes the objective function. The subsequent generation is created by recombining pairs of chromosomes in the current generation. Recombination between two chromosomes is the method through which the populations "evolve" better solutions. The solutions are probabilistically chosen for recombination based upon their fitness. Although the chromosomes which optimize the objective function well will have a higher probability of being selected for recombination than those which do not, they are not guaranteed to appear in the next generation. The "children" chromosomes produced by the genetic recombination are not necessarily better than their "parent" chromosomes. Nevertheless, because of the selective pressure applied through a number of generations, the overall trend is toward better chromosomes.

In order to optimize functions, genetic diversity must be maintained. When diversity is lost, it is possible for the GA to settle into a local optimum. There are two fundamental mechanisms which the basic GA uses to maintain diversity. The first, mentioned above, is a probabilistic scheme of selecting which chromosomes to recombine. This insures that schemata, other than the ones represented in the best chromosomes, appears in the subsequent generation. Only recombining good chromosomes will very quickly converge the population without extensive exploration, thereby increasing the possibility of finding only a local optimum. The second mechanism is mutations; mutations are used to help preserve diversity and to escape from local optima. The mutation operator is usually implemented as a random bit flip. In the traditional genetic algorithm, the mutation rate is kept at a very low constant.

The genetic algorithm is typically allowed to continue for an arbitrary number of generations. At the conclusion of the number of specified generations, the best chromosome in the final population, or the best chromosome ever found, is returned.

Unlike the majority of other optimization heuristics, genetic algorithms do not work from a single point in the function space. Methods which only use a single point to explore the function space are very susceptible to local optima. GAs continually maintain a population of points from which the function space is explored. This aids in searching multidimensional search space, in which many variables must be optimized, and in locating global optima.

1.2. Punctuated Equilibria

Currently, a large amount of research is being done towards understanding the capabilities of parallel genetic algorithms (pGA). A pGA is more than a means to improve the speed of the GA. Although there are many simple methods of increasing the speed of a GA, e.g. performing crossover, mutation, and evaluations in parallel, a pGA does not have to be executed on a parallel machine. A pGA is based upon the theory of punctuated equilibria.

In the paper *Distributed Genetic Algorithms for the Floor Plan Design Problem*, Cohoon et. al describe the theory of punctuated equilibria [Cohoon, 1988]:

> Punctuated Equilibria is based upon two principles: allopatric speciation and stasis. Allopatric speciation involves the rapid evolution of new species after a small set of members of species, peripheral isolates, becomes segregated into a new environment. Stasis, or stability, of a species, is simply the notion of lack of change. It implies that after equilibria is reached in an environment, there is very little drift away from the genetic composition of species. ... Punctuated Equilibria stresses that a powerful method for generating new species is to thrust an old species into a new environment, where change is beneficial and rewarded. For this reason we should expect a genetic algorithm approach based upon punctuated equilibria to perform better than the typical single environment scheme.

The implication of this upon the structure of the genetic algorithm is that given a single large population, the population will eventually converge to an equilibrium. The children chromosomes produced thereafter will be very similar to each other and to their parents, thereby rendering crossover operators largely ineffective. One method of resolving this problem is to create separate subpopulations. Each subpopulation evolves its chromosomes independently from all other subpopulations. The fitness used to determine probability of selection is relative only to the other members within the subpopulation. Independent evolution of subpopulations should yield closely competitive, yet possibly unique results in each subpopulation. In order to continue evolution after the subpopulations have converged, members of species from outside subpopulations can periodically be introduced.

There are at least two uses for parallel subpopulations. The first is, as mentioned above, to preserve diversity and to ensure perpetual novelty in the population's "gene pool". Parallel subpopulations have shown their effectiveness by solving many "GA-hard" problems which other GAs were not able to solve [Whitley & Starkweather, 1990], [Liepins & Baluja 1991]. The second use of the subpopulation structure is to emphasize different characteristics in the chromosomes. For example, in multi-objective functions, the evaluations in each subpopulation can be used to emphasize a different objective. When members of subpopulations are mixed, the genetic information may be combined to reveal chromosomes which are strong with respect to more than a single objective. A multi-objective optimization with parallel subpopulations can be found in [Husbands, 1991].

1.3. Implementing Punctuated Equilibria

It is not necessary to have a parallel machine to implement parallel subpopulations. However, it is important that each subpopulation has evolved a number of generations before individual chromosomes are swapped between subpopulations. This can easily be implemented serially. However, for the remainder of this section, it shall be assumed that the implementation is on a multi-processor system.

Each subpopulation has a dedicated processor. A set of chromosomes (n) is assigned to each processor. In the model used here, there are (N) processors. Therefore, the total number of chromosomes, is n x N. There are E epochs, or major iterations. During each epoch, every processor works in parallel, yet independently, evolving its chromosomes. There are G generations per epoch. After each epoch has ended, a small sample of chromosomes from each processor is swapped. Then, the next epoch is started. The analogy to punctuated equilibria is that after G generations, the chromosomes will have slowed down their evolution by coming to an equilibrium within their own subpopulation. Although this may not always be the case, given a sufficiently large G, the chromosomes will become similar. With the introduction of new chromosomes from another subpopulation, it should be possible, through many generations, to incorporate the new schemata into the current population. By controlling the number of chromosomes that are swapped, it is possible to control the amount of potentially new genetic information that is introduced into each subpopulation. More details of the specific implementation issues and parameters used will be given in section 3.

In order to ensure complete mixing of chromosomes throughout all subpopulations, the swaps are not always between the same subpopulations. One method of implementing the swapping procedure is to skip e subpopulations between the source and the destination subpopulation, where e is the number of epochs passed, [Whitley & Starkweather, 1990] see Figure 1. Although the sudden injection of new material is an important aspect of simulating punctuated equilibria, it is not always effective. Many times the population into which new material is introduced is already entirely settled into an equilibrium state. If this is the case, it is possible that the new information may not be incorporated as it is so dissimilar to the existing information.

Other parallel schemes involve swapping only with the closest neighbors. Alone, neither this nor the previously mentioned scheme addresses the problem of a subpopulation's potential resistance to change. One of the approaches used to address this problem is to create a separate subpopulation to which all the chromosomes are swapped. Although this aids in avoiding the problem of resistance to change as a predominant schemata structure does not already exist in the newly created subpopulation, its potential has not yet been fully explored. [Liepins & Baluja, 1991].

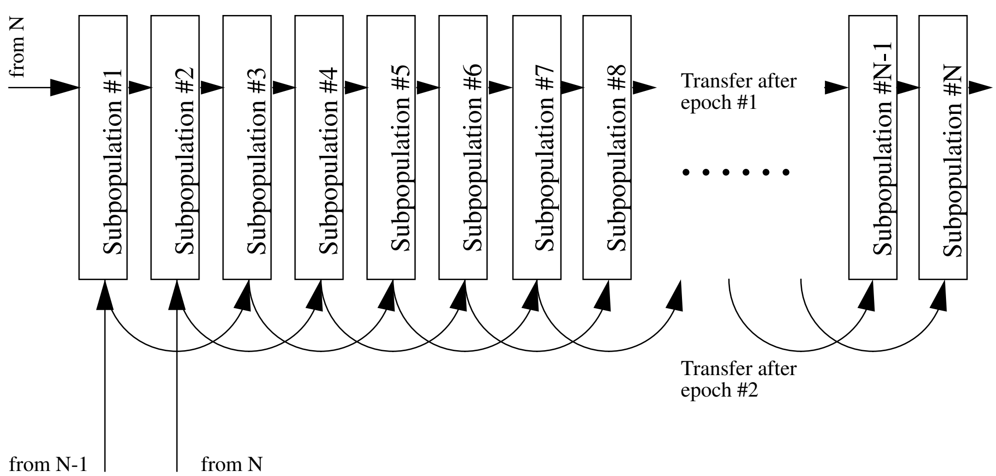

Figure 1. Typical transfer of chromosomes between subpopulations in parallel genetic algorithms. Transfer after epoch 1 and 2 shown.

In the next section, another type of parallelism for the GA is discussed. It incorporates the theory of punctuated equilibria, but attempts to avoid the sudden introduction of chromosomes into subpopulations by using overlapping population boundaries.

2. Massively Distributed Parallel Genetic Algorithms: an Overview

In order to address the problems associated with the sudden introduction of genetic information, the basic architecture of the parallel genetic algorithm has been modified. The massively distributed parallel genetic algorithm works upon the premise that although separate populations do yield a benefit, by reducing the severity of the boundaries between subpopulations, it might be possible to overcome the problems associated with sudden introduction of new material. One way in which to view this modified form of parallelism is to conceptualize the populations as overlapping, see Figure 2. This structure allows for the gradual transfer of genetic information without the sudden introduction of chromosomes at set intervals. The swapping portion of the parallel genetic algorithms is inherent to the structure, as genetic information “flows” through the network of GAs.

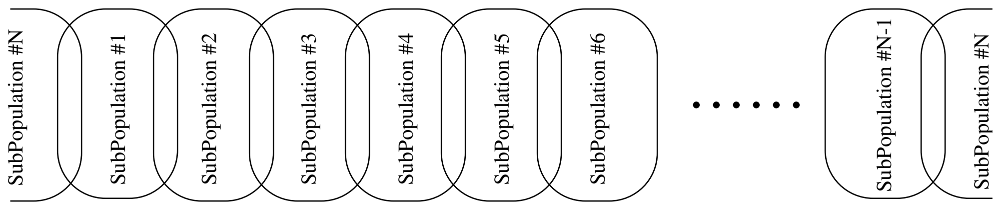

Figure 2. Overlapping Populations in a Massively Distributed Parallel Genetic Algorithm.

The motivation behind this subpopulation organization is that subpopulations which are a large distance apart (relative to N), will evolve comparatively unique chromosomes in a manner similar to simple, disjoint, parallel subpopulations. However, depending upon the degree and structure of the overlap of the populations, all of the subpopulations potentially have a bearing upon each subpopulation’s evolution. Of course, the subpopulations within a close locality to each other will have a much greater influence on each other than those a large distance apart. As with standard parallel GAs, the larger the number of subpopulations, the greater the potential diversity in evolutions.

The danger of a suboptimal state being reached in the mdpGA is greater than in the structure employed by traditional pGAs for two reasons. The first is that the mdpGA employes a greater degree of swapping between subpopulations than pGAs. Secondly, in the implementations tested, there are significantly fewer chromosomes per population in the mdpGA than in traditional pGAs. In order to address this problem, the mdpGA relies upon the size of the network of genetic algorithms and the controlled degree of overlap to allow unique evolutions in different portions of the network.

One of the issues raised in creating this type of genetic algorithm is determining the extent of overlap between subpopulations, and with which other subpopulations the overlap should be constructed. For example, if each subpopulation overlaps only 2 others, the good chromosomes would “flow” from processor to processor very slowly. If the chromosomes overlapped quite a few other subpopulations, the good chromosomes would rapidly flow throughout the GA structure. However, this may lose the advantages of punctuated equilibria. Many different issues regarding topology need to be addressed, such as should the populations be connected as described above, in a linear manner, or should the overlapping populations be virtual, almost simulating neural network connections. Furthermore, should the connections between subpopulations be fixed, or time varying? These issues are examined in the section 3.2 and again addressed in the conclusions.

3. The mdpGA: Implementations & Details

As stated before with reference to traditional parallel genetic algorithms, it is not necessary to implement the mdpGA on a parallel machine. Simulations of parallel architectures for this GA can be done successfully on a serial machine. However, the discussion which follows assumes the use of the MasPar MP-1, described in the next section. The implementation specifics of the mdpGA, described in section 3.2, map very easily to the capabilities of this machine.

3.1. Machine Specifics

The hardware used to perform the test runs was the MasPar MP-1. This is a massively parallel computer, consisting of an Array Control Unit (ACU) and a two dimensional array of processing elements (PE). The array size is 64 x 64, for a total of 4096 processing elements. The ACU has 1 Mbyte of RAM with up to 4 Gbyte of virtual instruction memory. The ACU is used to control the PE array and to perform instruction on data which is not located on the PE's dedicated memory. Each PE has 16 Kbytes of dedicated RAM. The MasPar is a Single Instruction Multiple Data (SIMD) machine. Put simply, this means that each PE receives the same instruction simultaneously from the ACU. Through control statements, processors can be either active or non-active. The PEs which are members of the active set will execute the instruction they receive from the ACU*, on local data*. Those which are not members of the active set will do nothing.

The processors can be accessed either through relative addressing mode (from other processors) or absolute addressing. Further, they have two modes of numbering for identification. They can either be accessed by x,y coordinates or by row-major ordering. This allows an efficient programming model for the three implementations described in the next section.

The programming language used was the MasPar Parallel Application Language (MPL). This is an extended set of Kernighan and Ritchie C with extra directives for controlling the PE array and parallel data structures.

3.2. Population Architecture

The mdpGA was tested with three different implementations. The first implementation assumes a circularly linked linear ordering of processors as shown in Figure 2. Each processor evolves only two chromosomes per generation. These two chromosomes are chosen from a population of 10 chromosomes. The population of 10 is comprised of 1 chromosome from each of the four immediate left processors, 1 chromosome from each of the four immediate right processors and the two chromosomes which were evolved in the processor during the previous generation. Each of the chromosomes selected from neighboring processors is chosen randomly from the 2 evolved at the remote processor. The fitness of every chromosome in each population is calculated. The fitness is relative only to the other chromosomes in the population of 10. Two chromosomes from this set of 10 are probabilistically, based upon the relative fitness, chosen for recombination. The other 8 chromosomes which are not chosen for recombination are discarded. See Figure 3 for a pictorial explanation. In the next generation, the two "children" chromosomes produced (through crossover and mutation) are available for the recombination, either by the processor on which they are located, or by its neighbors.

The second implementation tested assumes a two dimensional array of processors. Similar to the previous implementation described, each of the processors evolves two chromosomes, which are chosen from a population of 10. The two dimensional array is toroidal. Instead of selecting the population members from only the 4 left and 4 right neighbors, the eight immediately neighboring processors donate to the population of 10. See Figure 4. The neighbors are located to the immediate E, NE, N, NW, W, SW, S, SE of the processor. The remainder of the procedure is completed in exactly the same manner as the previous implementation.

.

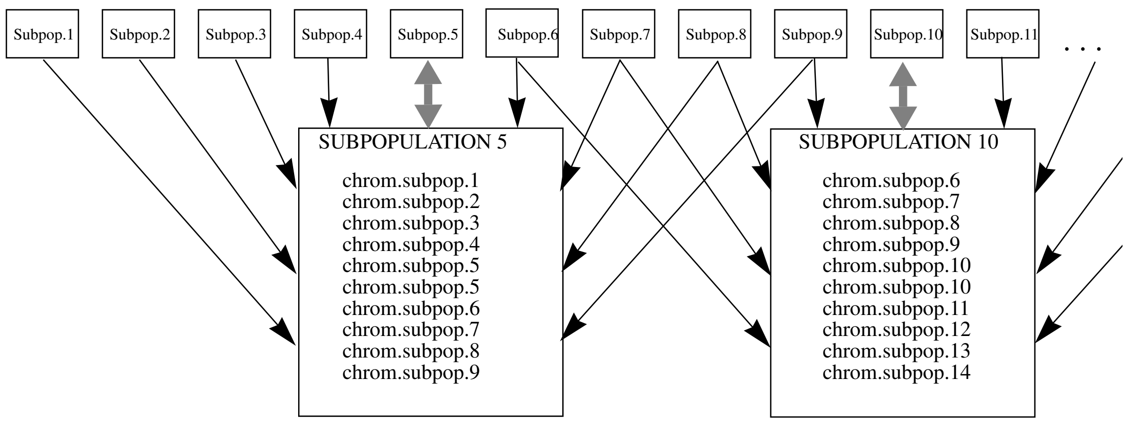

Figure 3. The architecture of subpopulations, arrangement #1. Subpopulation 5 and 10 are shown enlarged. They randomly select 1 chromosome from the 2 chromosomes in each of their 8 closest neighbors, 4 from the left neighbors 4 neighbors, 4 from the right. They also incorporate their own two chromosomes into the population. From this population two chromosomes are probabilistically chosen for breeding. The 8 chromosomes not selected are discarded.

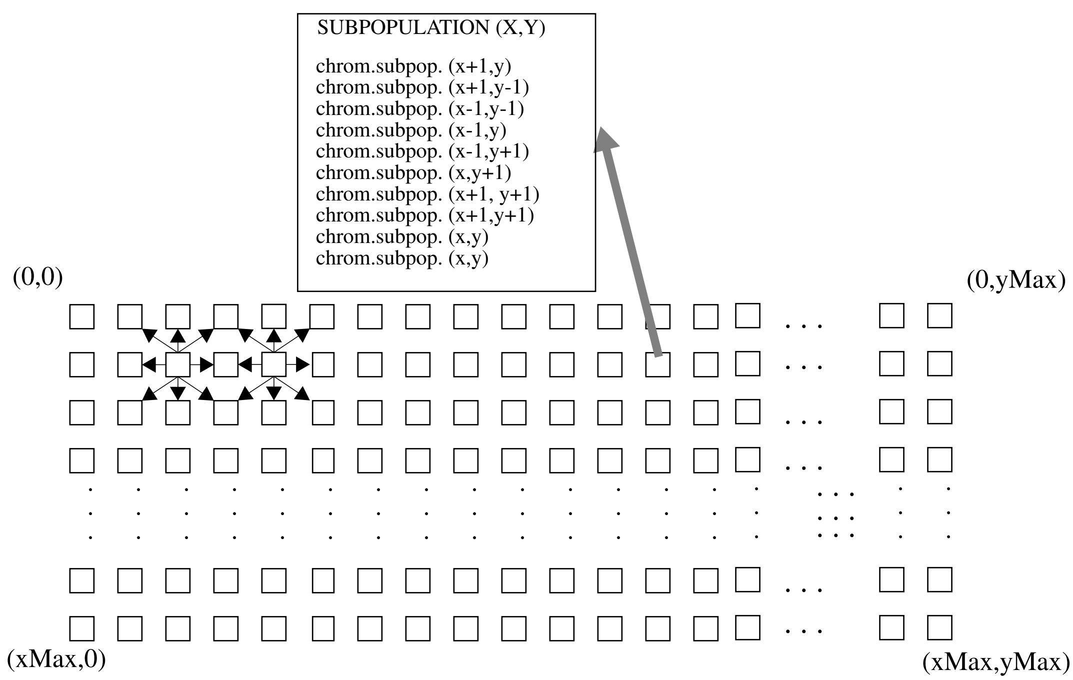

Figure 4. The architecture of subpopulations, arrangement #2. A two dimensional array of processors is assumed. Each processor contributes one of its two chromosomes to each of its 8 nearest neighbors. A sample population for processor (x,y) is shown. The processors are connected in a toroid.

In the third implementation, a linear ordering is once again assumed. One chromosome from each of the 3 immediate left processors and one chromosomes from the 4,5,6 processors from the right are used in the candidate population. See Figure 5. Unlike the first implementation, the immediate right processors are not used. The 4 remaining positions, out of the population of 10, are filled with 2 copies of each of the chromosomes evolved in the previous generation. Unlike the other subpopulation architectures, which include 8 neighbors, this only includes 6. This is done to examine the effect of reducing the spread rate of chromosomes upon performance. Once the population of 10 chromosomes is selected, the mpdGA progresses in the same manner as described earlier. Once again, each processor only evolves 2 chromosomes per generation. This model was chosen because it allows a faster spread of good chromosomes than the first implementation and a slower spread than the second implementation. See Figure 6.

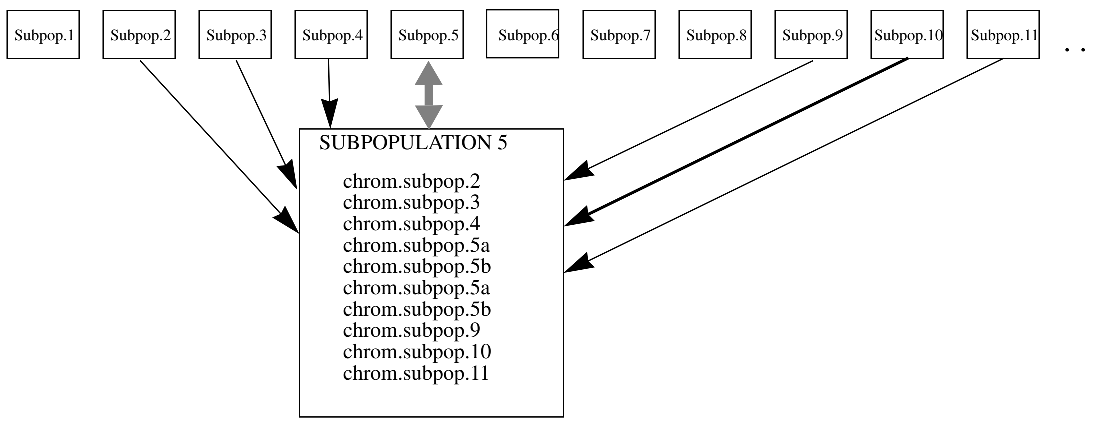

Figure 5. The architecture of subpopulations, arrangement #3. Subpopulation 5 is shown enlarged.

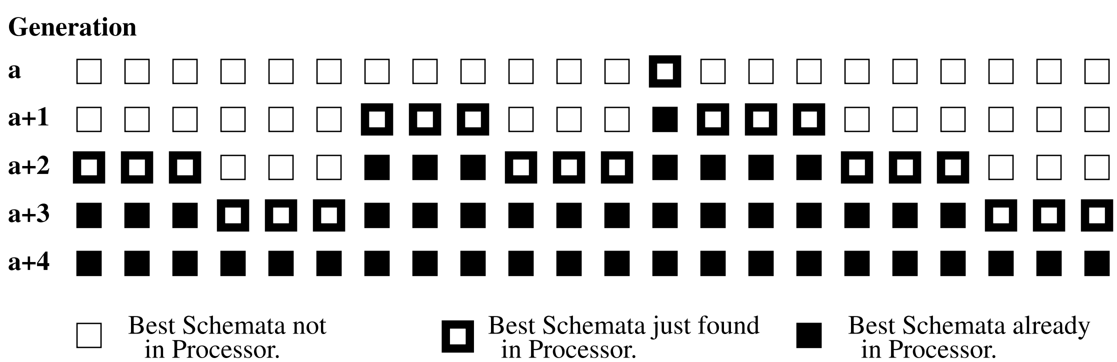

Figure 6. Making the large assumption that the best schemata is not lost during crossover or mutation, the above diagram depicts how the schemata could spread through the processors, using mdpGA configuration #3.

Making the large assumption that the best schemata is not lost during crossover and mutation, the maximum spread rate of a very good chromosome is different for each of the implementations. In arrangement #1, the linear ordering, in which the 4 processors to the left and 4 to the right see the best chromosome immediately, assuming 4096 processors arranged in a linear ordering, the lower bound on the number of generations for all of the processors to see the best chromosome is 512-1, 511 generations. In arrangement #2, the square array of processors, the minimum number of generations to get from one processor to any other is 1/2 the diagonal of the square. Assuming a 64 * 64 processor arrangement, within approximately (32 -1) 31 generations, the chromosome could be seen by all of the processors. Using arrangement #3, a compromise between the first two, with regard to speed, the number of generations is 455. All of the -1 factors arise because in the first generation that a good chromosome is found, it is immediately included in its neighbors selection of 10 chromosomes. These estimates are only on the lower bound of the spread of the best chromosome. The chromosome will not generally spread this fast as in order for this speed to be achieved, the chromosome must be reselected for recombination at each generation, and none of the valuable schemata can be destroyed by crossover and mutation. Further, this also assumes that no better chromosomes are found, and that a chromosome of the same evaluation is not found without first "seeing" the original chromosome with the evaluation. These issues are discussed again in section 4.3. Another topology, termed the "ladder" population structure, has been explored by Muhlenbein and Schleuter,[Muhlenbein,1989], [Scheduler, 1990].

3.3. Implementation Specifics

For the test runs described here, the mdpGA used only standard GA mechanisms in order to allow a fair comparison of the capabilities of the mdpGA with other GAs.

3.3.1. The Recombination Operator

The recombination operator used for testing this algorithm is the two point crossover. This is a very general crossover operator that has been shown to be effective in GA literature. See Figure 7 for details. For a discussion of the effectiveness of two point crossover, as compared to other multi point crossover operators, the reader is referred to [De Jong, 1990], as compared to the uniform crossover operator, the reader is referred to [Syswerda, 1990].

TWO POINT CROSSOVER

SELECT TWO CHROMOSOMES FOR RECOMBINATION
chromosome A: 01001001
chromosome B: 10011011

CHOOSE TWO POINTS AT RANDOM (>= 0 and <= length of chromosome)
point a: 4 point b: 7

SWITCH CONTENTS OF CHROMOSOMES BETWEEN TWO POINTS

```
010|0100|1 AFTER CROSSOVER: 01011011
100|1101|1                  10001001
```

Figure 7. A brief description of two point crossover.

3.3.2. The Mutation Rate

Mutation is implemented as a bit flip which occurs in the "children" chromosomes after the two point crossover has taken place with the parents, and before the chromosome is evaluated. The mutation rate is kept at a constant 1%. Although adaptive mutation rates have empirically shown to greatly aid in maintaining diversity [Whitley & Starkweather, 90], [Liepins & Baluja, 91], [Cobb, 90], the effectiveness of the massively distributed approach should first be measured without the introduction of other non-standard factors.

3.3.3. Elitist Selection

This is a commonly employed tool to ensure that the progress made by a GA is not lost due to random chance. Because a GA's selection of parent chromosomes is probabilistic, it is not guaranteed that the best chromosome in a particular generation will survive to the subsequent generation. It is also possible that if the chromosome is selected for recombination, some of the good genetic material may not survive through the crossover and mutation operators. A modest form of elitist selection is used to address this problem. Elitist selection carries the best chromosome from the population of 10 candidates for recombination, from generation g to generation g+1. This does not imply that the best chromosome will be selected for recombination. Rather, it means that the chromosome will be in the population of 10 which are candidates for recombination. Because the number of population spaces is limited to 10, the best chromosome from the previous generation replaces the worst chromosome in the current generation. The worst chromosome is equated with the chromosome with the worst relative fitness. The drawbacks of this strategy is that it can be detrimental when the GA is caught in a local optima, as the elitist selection may preserve the local optima in the population's candidates for recombination.

3.3.4. Initial Selection of Chromosomes

The initial selection of chromosomes is random. Each processor is given two chromosomes to recombine, mutate and then evaluate. From here, the process of selecting the neighbor's chromosomes etc. is started. The majority of representations for chromosomes that have appeared in GA literature have been as binary bit strings. The same representation is used in these tests. The number of 0's and 1's should be approximately equal, with random placement within the chromosome.

4. Test Problems

4.1. Problem Descriptions

Six classes of problems were empirically tested, DeJong's five function test suite, three subset-sum problems, two orderings of a partially deceptive order 4 problem, two orderings of fully deceptive order 4 problems, 2 sizes of gap problem and three versions of all ones problems.

4.1.1. DeJong's Test Suite

DeJong's test suite is comprised of five minimization problems used to test the effectiveness of GAs [DeJong, 1975]. The test suite was designed to incorporate functions with the following characteristics: continuous/discontinuous, convex/nonconvex, unimodal/multimodal, quadratic/nonquadratic, low dimensionality/high dimensionality, and deterministic/stochastic. [Goldberg, 1989]. The functions in the test suite were coded using the standard binary counting system. Other implementations, such as gray code, were also considered. Although gray code has shown to work significantly better than standard binary coding, [Caruana & Schaffer, 1988], it was not used in order to allow evaluation of the effectiveness of the mdpGA without the bias of outside factors.

The functions are:

Function #1:

$$f1\,(x_1, x_2, x_3) \;=\; \sum_{j=1}^{3} x_j^2$$

$$-5.12 \le x_i \le 5.12,\; i \;=\; 1, 3$$

Function #2:

$$f2\,(x_1, x_2) \;=\; 100\,(x_1^2 - x_2)^2 + (1 - x_1)^2$$

$$-2.048 \le x_i \le 2.048,\; i \;=\; 1, 2$$

Function #3:

$$f3\,(x_1, \ldots, x_5) \;=\; 30.0 + \sum_{j=1}^{5} \lfloor x_j \rfloor$$

$$-5.12 \le x_i \le 5.12,\; i \;=\; 1, 5$$

Function #4:

$$f4\,(x_1, \ldots, x_{30}) \;=\; \sum_{j=1}^{30} x_j^4 + Gaussian\,(0, 1)$$

$$-1.28 \le x_i \le 1.28,\; i \;=\; 1, 30$$

Function #5:

$$f5\,(x_1, x_2) = \frac{1}{\dfrac{1}{500} + \displaystyle\sum_{j=1}^{25} \frac{1}{j + \displaystyle\sum_{i=1}^{2} (x_i - a_{ij})^6}}$$

$$a_{ij} = \{-32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32\}$$

$$a_{ij} = \{-32, -32, -32, -32, -32, -16, -16, -16, -16, -16, 0, 0, 0, 0, 0, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32\}$$

$$-65.536 \le x_i \le 65.536,\ i = 1, 2$$

4.1.2. Subset-Sum

The subset sum problems are NP-Complete. The problems can be stated as follows: given S elements, each of a possibly unique weight, is there a subset of S that adds up exactly to an arbitrary number, T? This problem was implemented as a 120 bit chromosome. Each bit represented a unique object. The object was assigned a random weight between 1 & 200. The weight T was selected to be either 1/4, 1/20, or 1/40 of the sum of the weights of the objects. The only addition to the problem was that it was insured that the sum of the weights was divisible by 4, 20 & 40, respectively. Note that this does not imply that a subset was guaranteed to sum to the amount desired. Because the initial chromosomes were started with approximately 50% distribution of 0s and 1s, and the weights selected were selected by a uniform distribution between 1 & 200, finding a set to sum to 1/2 of the total sum usually takes less time than finding set to sum to 1/3 the total sum. The reason is that given a random distribution of 0 & 1s, and a random distribution of weights, the total weight associated with the 0 bits and the total weight associated with the 1 bits are very similar.

4.1.3. Partially Deceptive Order 4

The partially deceptive problem is a 40 bit long maximization problem. It is comprised of 10 sub problems, each 4 bits long. The subproblems are evaluated using the following lookup table.

CHROMOSOME	EVALUATION	CHROMOSOME	EVALUATION
1111	16	0110	14
0000	28	1001	12
0001	26	1010	10
0010	24	1100	08
0100	22	1110	06
1000	20	1101	04
0011	18	1011	02
0101	30	0111	00

Figure 8. 4 Bit Evaluations of a partially deceptive function.

The problem was attempted using two different ordering of bits. The first encoding is block encoding, the placement of the 4 bits which comprise a subproblem are located next to each other. The second encoding is interleaved. With the use of two point crossover, the first encoding is much easier for the GA to solve than the second. The encodings are shown in Figure 9.

Block Encoding:	aaaabbbbccccddddeeeeffffgggghhhhiiiijjjj
Interleaved:	abcdefghijabcdefghijabcdefghijabcdefghij

Figure 9. Two encodings of the order 4 deceptive problem & partially deceptive problems.

4.1.4. Fully Deceptive Order 4

This is the order 4 deceptive problem defined by Whitley and Starkweather, in their paper GENITOR II [Whitley and Starkweather, 1990]. The problem is a 40 bit long maximization problem, and is comprised of 10 sub problems, each 4 bits long. The subproblems evaluate 4 bits using the following lookup table, Figure 10.

CHROMOSOME	EVALUATION	CHROMOSOME	EVALUATION
1111	30	0110	14
0000	28	1001	12
0001	26	1010	10
0010	24	1100	08
0100	22	1110	06
1000	20	1101	04
0011	18	1011	02
0101	16	0111	00

Figure 10. 4 Bit evaluations of a fully deceptive function.

The same two encodings which were used in the partially deceptive problems are used in this problem. See Figure 9 for details. It should be noted that this problem is significantly more difficult for a GA to solve than the previously described partially deceptive problem as the penultimate optimal and optimal solutions have a hamming distance the size of the problem. In the partially deceptive problem, the penultimate optimal and optimal solutions had a hamming distance 1/2 the size of the problem.

4.1.5. The Gap Problem

The Gap Problem is a maximization problem. The gap function f(x), with gap of size Y, starting at point P, with o(x) being the number of ones in the bit string, is an order function defined by (see Figure 11):

$$f(x) = \begin{cases} 2P + Y - o(x) - 1, if(P \le o(x) \le P + Y - 1) \\ o(x), if((o(x) < P) \vee (o(x) > P + Y - 1)) \end{cases}$$

This problem was tested on a 120 bit chromosome string. Gap sizes of 20 & 25 were tried with a starting point, P=60, 1/2 of the size of the chromosome. Populations were initialized, as in all the other cases, with a random distribution of 0's and 1's. It is expected that the majority of the chromosomes will start with approximately 60 ones, therefore they must overcome the gap immediately. See Figure 11 for the evaluations per number of ones in the chromosome.

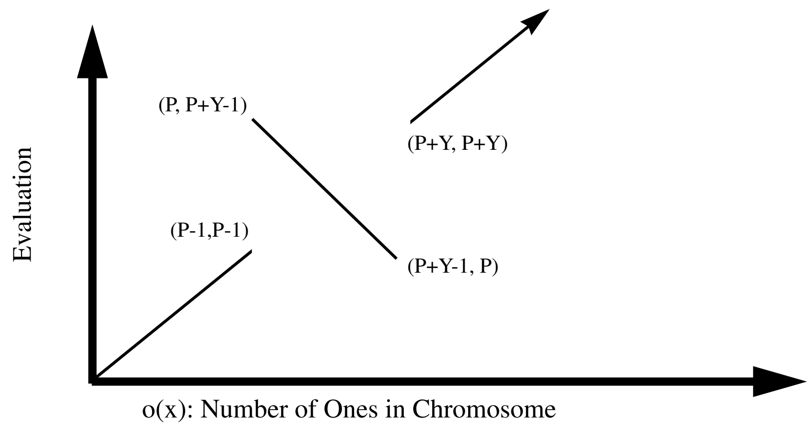

Figure 11. The Gap Problem. o(x) is the number of ones per chromosome. The gap size is Y. The starting point of the gap is P.

4.1.6. Three All Ones Problems

Three versions of the all-ones problem were tried. The first version was the straight all ones problem. The objective of this problem is to find the chromosome which contains a 1 in each bit position. The optimal answer to this problem is just the size of the chromosome. The chromosome length tested was 120 bits.

The second version of the all-ones problem contains bits which are meaningless. This problem was encoded as a 180 bit problem, but only the first 120 bits were counted toward the evaluation. The optimal solution to this problem is at 120.

The third version of this class of problem is slightly different than the previous two, it is the contiguous bits problem. The optimal solution to this problem is also a chromosome which contains all ones. However, in evaluating the chromosome, points are only given for 1s which also have at least one other neighbor which has a value of 1. If there exists a 1 which has zeros as its two neighbors, no points are given for the bit.

4.2. Algorithms Tested

Four algorithms were tested, the three implementations of the mdpGA described in section 3.2 and a 40 sub-population parallel genetic algorithm. The two implementations of the mdpGA worked significantly faster than the pGA on almost all problems, the details of the timings can be seen found in [Baluja, 1992].

The pGA was loosely based upon the pGA described in [Whitley & Starkweather, 1990]. It was run with a 1% mutation rate, and two point crossover (the mdpGA was tested with the same mutation rate and crossover operator). The pGA also employed modest elitist selection, in which the single best chromosome from each generation automatically was carried to the subsequent generation. Similar to the mdpGA, the elitist selection does not imply that the chromosome will be selected for recombination, only that it is a candidate for selection. The best chromosome replaces the worst chromosome in the subsequent generation. Each population had 100 chromosomes, for a total of (40 * 100) 4000 chromosomes evaluated simultaneously.

There was a very small amount of communication between the subpopulations. Assuming a circular ordering of subpopulations, after an epoch of e=100 generations, the best chromosome from one population migrated to a subpopulation e subpopulations away. “e” is defined to be the number of epochs that have passed. As the population was a set size, the migrating chromosome replaced the worst chromosome in the target subpopulation.

When comparing results, it should be considered that the mdpGA evaluates 8192 chromosomes per generation, and the pGA evaluates only 4000.

4.3. Results & Discussion

The results are shown for the 17 test problems in Figure 12. They are the average of 10 runs per problem for each algorithm. The maximum number of allowed generations for the pGA is 3000, after 3000 the attempt is considered a failure. The maximum number of generations before failure for each of the mdpGAs is 1400.

The mdpGA has done well in these test cases. One problem which stands out, in particular, is the simple all ones problem. The all ones problem using the three versions of the mdpGA averaged 114.0, 90.5, and 107.7 generations. However, using a simple pGA, the average generation to find the optimal was 648.2, a very large increase. Perhaps one of the reasons for this is in the mdpGA, good chromosomes rapidly spread through the entire population. Depending upon the implementation of the overlap between populations, a large portion of the population has access to the best chromosome very shortly after it is found. A sample run, using the 2D array configuration, shown in Figure 14, displays the number of processors which “have seen” the best chromosome found in each generation. The term “has seen” does not imply that the processors are currently recombining the best chromosome, rather that it is available, with the aid of elitist selection, in their population of 10 candidate chromosomes. The sudden drops in the number of processors represent generations in which a “new” best chromosome is found.

The second implementation of the mdpGA, with the population overlapping with the 8 closest neighbors, allows a good chromosome to be immediately “taken” by 8 processors as soon as it is found. However, for it to get any further than these 9 processors is more difficult. In order for more than the original 9 processors to take the chromosome, it must again be selected for recombination. Assuming that it is selected, valuable schemata must not be destroyed by crossover or mutation operators. This is true because although the processors which surround the original 9 will incorporate the resultant chromosomes into their population, they must select them for recombination based upon their evaluation, which may not be as good as their parents. Further, if the crossover and mutation have destroyed valuable schemata, the children produced will not be preserved in processors by elitist selection unless the evaluations are better than any the processor has seen thus far. On the other hand, if the important schemata is only a small part of the total chromosome, the chances of the chromosomes being spread throughout the network with valuable schemata intact is much greater. The reason for this is that the crossover and mutation operators have a smaller chance to destroy the valuable schemata. Therefore, the children chromosomes, which may differ from their parents, may do so in inconsequential ways.

Test Function	**mdpGA**	**mdpGA**	**mdpGA**	**pGA**
	4096 Linear Order	*64 *64 2D Array*	*4096 Linear With Skip*	*40 Subpopulations*
DeJong Function #1	32.0	29.8	30.6	79.0
DeJong Function #2	40.0	38.6	43.9	111.8
DeJong Function #3*	22.0	19.5	20.4	64.5
DeJong Function #4	See Figure 13 for details.			
DeJong Function #5**	17.9	18.0	17.8	18.0
Subset Sum (1/4)	21.0	14.0	12.0	35.6
Subset Sum (1/20)	68.0	55.0	65.0	344.5
Subset Sum (1/40)	95.4	76.8	87.8	629.0
Partially Deceptive (Block)	39.0	32.0	38.0	95.1
Partially Deceptive (Interleaved)	70.0	53.0	75.0	252.5
Deceptive Order 4 (Block)	90.0	57.5	78.4	305.9
Deceptive Order 4 (Interleaved)	1220(4)	742.5	942.2(5)	1634.7
Gap Problem (Gap Size 20)	161.2	126.3	164.3	675.9
Gap Problem (Gap Size 25)	816.4(5)	441.2	699.1(9)	776.0
All Ones Problem	114.0	90.5	107.7	648.2
Sparse All Ones Problem	134.0	94.8	113.3	342.0***
Contiguous Bits Problem	131.0	90.8	111.0	609.1

Figure 12. Results for the 17 test problems. Each entry represents the average number of generations it took to find the optimal solution, in the cases in which it was found. When comparing average number of generations, it should be remembered that the mdpGA evaluated 8192 chromosomes per generation, while the pGA evaluated only 4000. A number in parentheses indicates that the optimal solution was only found the specified number of times, out of 10.

* The stopping criterion for DeJong's F3 was an evaluation of -30.

** The stopping criterion for DeJong's F5 was an evaluation of 0.998004.

*** Due to memory restrictions, this problem was attempted with 90 significant bits, and 30 extra bits (pGA only). The mdpGA runs were full size (120 significant bits, 60 extra bits).

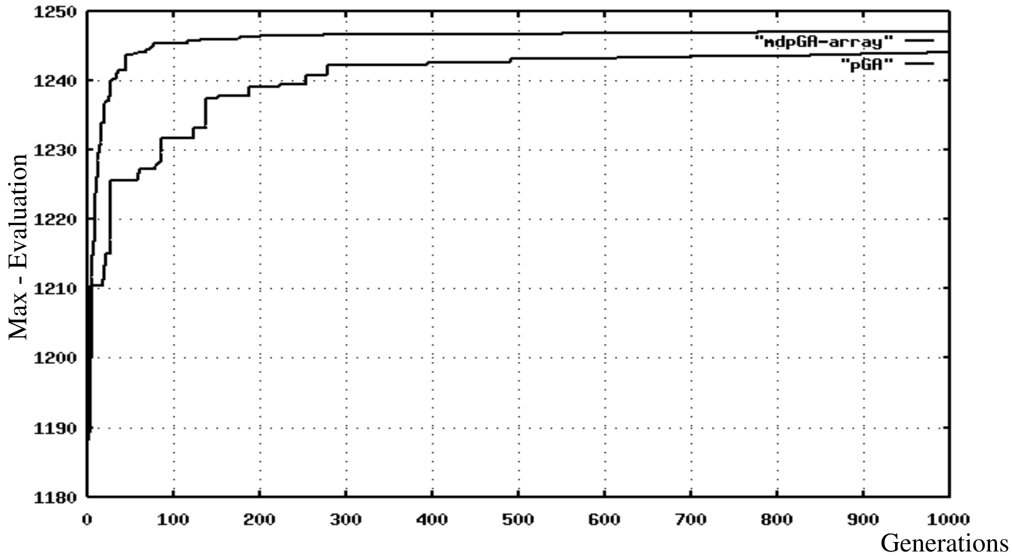

Figure 13. Sample run of DeJong's F4. For clarity only the mpdGA array structure and pGA are shown. Note that due to memory restrictions the pGA was run with 80 subpopulations with 50 chromosomes per population. The evaluations shown include the random gaussian factor.

The choice of how the subpopulations should overlap plays a significant role in how fast the chromosomes are spread through the population network. For certain classes of problems, it may be important to ensure that the flow of chromosomes is very slow, in order to allow for extremely different evaluations in different portions of the network. However, in other applications, a fast flow may yield good answers quickly as good chromosomes are rapidly spread through the population. The structure of the overlap is very important, as can be seen in the significantly different success rates of the mpdGA on the Deceptive - Order 4 problem.

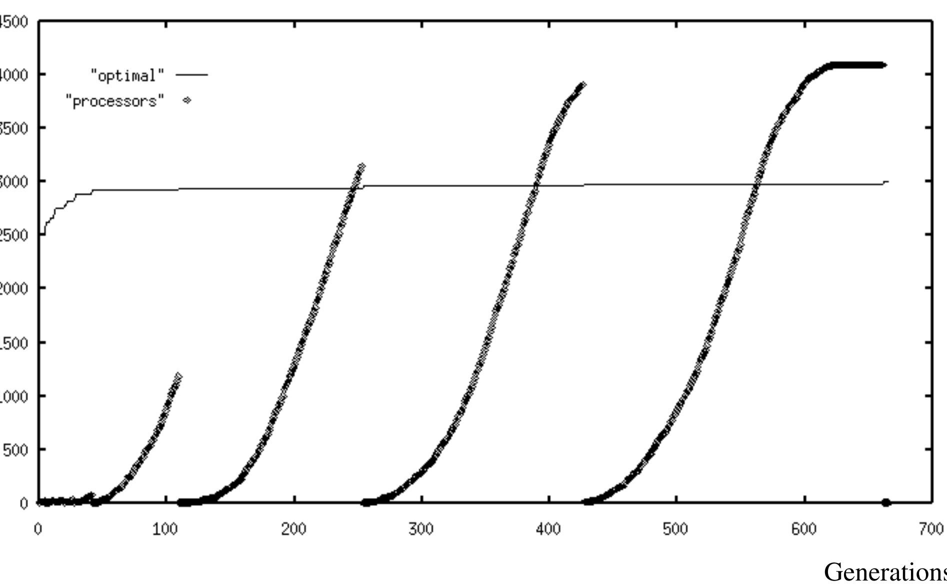

Figure 14. The Number of Processors which contain the best chromosome using the mdpGA to optimize the order 4 fully deceptive problem, interleaved. Also plotted is the best value * 10, for comparison. The sudden drops in the number of processors used, represent a new best solution found in one of the processors.

4.4. Summary & Future Research

Preliminary results on seventeen test problems have shown the mdpGA to be able to solve problems more efficiently than a simple parallel genetic algorithm. In order to further evaluate the worth of this structure, both harder test problems and different population topologies should be explored.

Perhaps the most pressing topic for future research is the need to design a topology which works well in a wide variety of problems. For the problems tested, the two dimensional array topology seemed to work the best. However, two future structures which would be interesting to examine would be one in which each processor is only connected to 1 of its nearest neighbors and a second in which the connections are made randomly, perhaps with a set maximum reaching distance.

One of the important applications of parallel genetic algorithms, which was not explored here, is the use of sets of populations to optimize different objectives in multi-objective functions. Each population evolves under the pressures of individual components of the complete problem. As stated in [Husbands, 1991] "...the solution to a complex problem is allowed to emerge from the simultaneous solution of a number of *simpler*, related sub-problems. Using this variation of divide and conquer, the inherent parallelism in a problem is brought out and thoroughly exploited." This method is directly applicable to the mdpGA; populations, on different processors, can work towards individual sub-goals. It would be interesting to determine the role that the position of the processor has on assigning objectives to each population.

Acknowledgments

I would like to thank Todd Jochem, Dean Pomerleau, Stephen Smith, and Chuck Thorpe for their helpful comments and suggestions regarding this paper.

5. References

Baluja, S. (1992) An Analysis of Genetic Algorithm Structure on the MasPar MP-1. Paper in Progress.

Caruana, R. and J. Schaffer (1988) Representation and Hidden Bias: Gray Vs. Binary Coding for Genetic Algorithms. *Proceedings of the 5th International Conference on Machine Learning*. Morgan Kaufmann. Los Altos. CA. June 1988 152-161

Cobb, H. (1990) An Investigation Into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having continuous, Time Dependent Nonstationary Environments. NCARAI Library. AIC-90-001.

Cohoon, J.P, S.U. Hedge, W.N. Martin & D. Richards (1988), Distributed Genetic Algorithms for the Floor Plan Design Problem. Technical Report TR-88-12. School of Engineering and Applied Science, Computer Science Department, University of Virginia.

DeJong, K.A. (1975) *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. (Doctoral dissertation, University of Michigan). Dissertation Abstracts International 36-10, 5140B.

DeJong, K.A. and W. Spears (1990) An Analysis of Multi-Point Crossover. NCARAI Library. AIC-90-014.

Eshelman, L. (1990). The CHC Adaptive Search Algorithm: How to have safe search when engaging in non-traditional genetic recombination. *Foundations of Genetic Algorithms*, Bloomington, IN..

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

Husbands, P., F. Mill & S.Warrington (1991), Genetic Algorithms, Production Plan Optimisation and Scheduling. *Parallel Problem Solving from Nature*, H.P. Schwefel & R. Manner Eds. Springer Verlag, Berlin.

Ingber, L and B. Rosen (1992) Genetic Algorithms and Very Fast Simulated Reannealing: A comparison. To be published in *Mathematical and Computer Modelling.*

Liepins, G.E. and S. Baluja (1991) apGA: an Adaptive Parallel Genetic Algorithm. Proceedings of *ORSA-TIMS CSTS Conference*, Williamsburg, Va. 1990.

Liepins, G. E. and M. D. Vose (1990). Representational Issues in Genetic Optimization, *Journal Expt. Theor. Artificial Intelligence*., 2, 101-115

Muhlenbein, H (1989) Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization. *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann. San Mateo, CA.

Schaffer, J.D., R.A. Caruana, L.J. Eschelman, and R. Das (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA.

Schleuter, M.G. (1990) , Explicit Parallelism of Genetic Algorithms through Population Structures. *Parallel Problem Solving from Nature*, H.P. Schwefel & R. Manner Eds. Springer Verlag, Berlin.

Syswerda, G. (1989) Uniform Crossover in Genetic Algorithms, *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann. San Mateo, CA.

Whitley, D. and T. Starkweather (1990). GENITOR II: a Distributed Genetic Algorithm, *Journal Expt. Theor. Artificial Intelligence*, 2, 189-214.