FEM Mesh Mapping to a SIMD Machine Using Genetic Algorithms

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

John S. Dunkelberg, Jr.

May 1996

Approved:

Dr. Lee A. Becker, Primary Advisor, CS Department, WPI

Dr. William R. Michalson, Co-Advisor, ECE Department, WPI

Dr. Robert E. Kinicki, Head of the Department of Computer Science

Abstract

The Finite Element Method is a computationally expensive method used to perform engineering analyses. By performing such computations on a parallel machine using a SIMD paradigm, these analyses' run time can be drastically reduced. However, the mapping of the FEM mesh elements to the SIMD machine processing elements is an NP-complete problem. This thesis examines the use of Genetic Algorithms as a search technique to find quality solutions to the mapping problem. A hill climbing algorithm is compared to a traditional genetic algorithm, as well as a "messy" genetic algorithm. The results and comparative advantages of these approaches are discussed.

Table of Contents

Table of Figures

Organization

This thesis presents an exploration of this specific mapping problem, using the genetic algorithm approach. To this end, the thesis is organized into seven sections. First it is necessary that we explain why the problem we are exploring is interesting and applicable to the real world, which we will do in the chapter titled Motivation. Then in the Background chapter, we will present the general introduction to the field. This is necessary in order to understand what we have done, why we have done it, and what our results have been. From that, we will proceed to a description of our exploration of the problem, in the Approach chapter. Having done some exploration of the problem, we will then explain how we have analyzed the results of this exploration, in Evaluation. From there we draw Conclusions as to the meaning and indications of our evaluations. Finally, we consider some Future Directions that research could take in tackling further aspects of this problem. The Appendices to this thesis provide bibliographical references as well as C++ code and results data.

Chapter 1: Motivation

As the engineering community has come to rely on CAD/CAE[1] for the design and analysis of high technology products, it has found a need for an automated method to calculate a variety of properties of the parts that make up these products. No practical engineer wants to work out the calculus, for instance, to determine the electromagnetic field around a highly irregularly shaped part consisting of a complex set of antennae and wave guides. Such an automated process does exist, through a technique called the Finite Element Method, hereafter referred to as FEM. Evaluation of real-world problems via FEM does, however, have a high computational cost. The time element of this cost can be reduced by employing a parallel computer. In this study we look particularly at a class of computer referred to as a SIMD[2] machine. To implement our FEM problem on a SIMD machine, we must solve the problem of how to map the topology of the FEM problem to the topology of the SIMD machine. To that end, we have studied the effectiveness of a class of computational algorithms known as genetic algorithms.

1.1: The Finite Element Method

The Finite Element Method works by decomposing the problem geometry into smaller, more regular geometric regions that can be attacked by simple mathematical models. Thus, as a first step, the problem is divided into a large number of regular two or three-dimensional geometric regions. A simple two-dimensional example of this method of subdivision of a problem is shown below (see Figures 1 - 3). Each of these elements can then be solved with a relatively simple set of equations. This process lends itself to conventional serial computational attack, being implementable as a highly iterated manipulation of large matrices. But FEM is an approximation, and one that becomes more accurate as the problem space is divided into smaller and smaller elements. As our target parts become more complex, and our need for accuracy increases, the cost in computational time increases greatly.

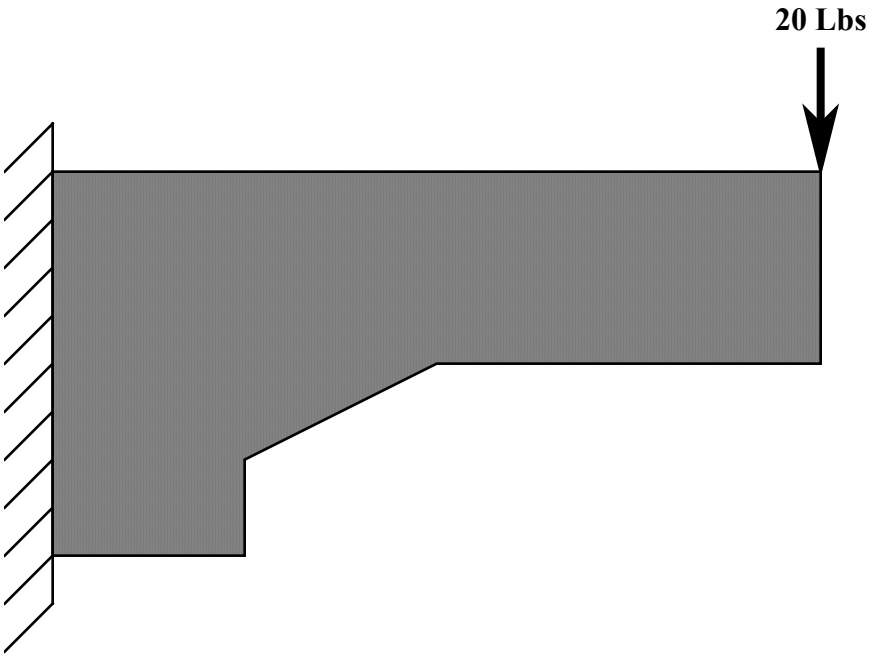

Figure 1: A cantilever beam problem

In this example, a force is applied to the end of a support member which is firmly affixed to an unmovable surface (referred to as "ground"). The cantilever beam problem is the basis of many mechanical stress analyses that might be solved using the Finite Element Method.

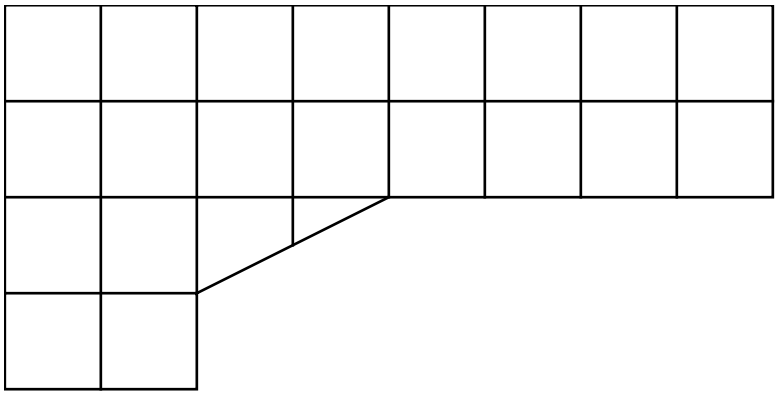

Figure 2: Subdivision of the problem into elements

The first step in solving such a problem is to break the target solid into a number of elements. Here we have broken the cantilever beam into 22 elements. In general, subdivision into smaller, more numerous elements provides a more accurate solution. However, the more elements a problem is comprised of, the longer it will take to solve.

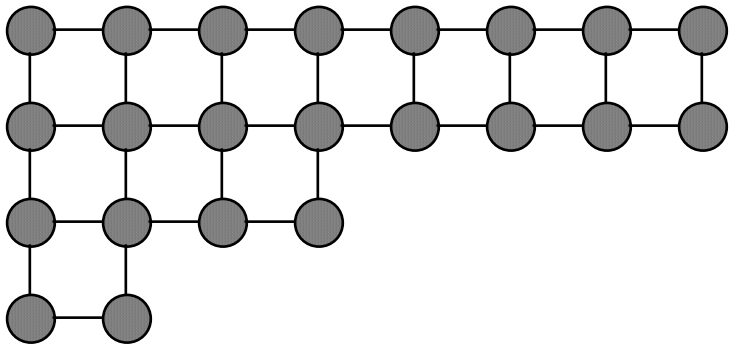

Figure 3: Neighboring nodes of the FEM model

The FEM model formulates a node at the center of each element. The nodes interact with each other along the element edges. In this case, the interaction is the application of force at our upper right-most node, which will convey that force onto its neighbors. To over-simplify the process, think of the force upon the upper, right-most node as a vector. Some of the magnitude of that vector is calculated to act upon that node, and a component of that vector is passed on to its neighbors. As we will discuss later, this communication between nodes is very similar to the methodology of a SIMD computer.

1.2: The SIMD parallel computer

Fortunately the Finite Element Method lends itself very well to parallelization.[3] Each node of the FEM mesh does the same set of operations, and passes the result of this operation on to its neighbor nodes. Thus, a potentially very powerful method by which to attack this problem is to run the analysis on a SIMD machine such as the MasPar MP-1[4], which has 1024 processors that are logically connected in a wrapped toroidal mesh. Each Processing Element (PE) of the

MasPar is connected by a direct (and low time-cost) communication link to its eight nearest neighbors, and by higher cost communication methods to any other given PE in the MasPar. This allows the computation of our FEM model, which is regular and repetitive in nature, to be quite elegantly solved on a SIMD machine.

A simple example of a Finite-Element problem on a SIMD machine can be shown using a thermal model. Consider a plate under thermal stress that can be modeled as a two dimensional brick 5 elements by 5 elements large. Along the left hand side, we apply a constant condition, a temperature of zero degrees. At the bottom left hand corner, we apply a constant conditions of one hundred degrees. Our computation for the change in temperature of our brick is very simple. At any time step *n*, the temperature of a given point will be given the average of its neighbors' temperatures at time *n-1*. Thus our initial condition can be represented as shown in Figure 4.

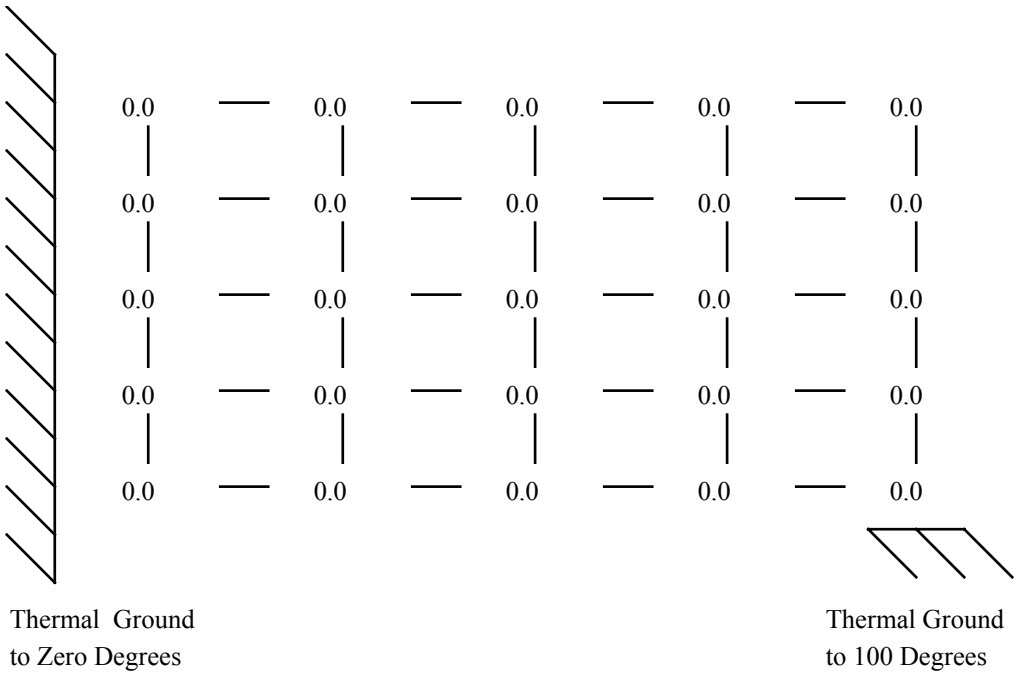

Figure 4: Initial Condition of Thermal FEM Model

We have initialized the model so that every processing element contains the initial condition, i.e. zero degrees. Consider a SIMD machine that is 7 by 7 processing elements, built on the same principles as the MasPar. We can embed our FEM problem inside of this machine as is shown in Figure 5. While each PE has the memory capacity to hold many datawords, here we only will use a floating point number and a flag in each PE. Our active problem elements are the central 5x5 PEs. In the border PEs we have stored a negative value, and where there is a constantly held condition we have stored a flag, represented in the figure by a 'C'.

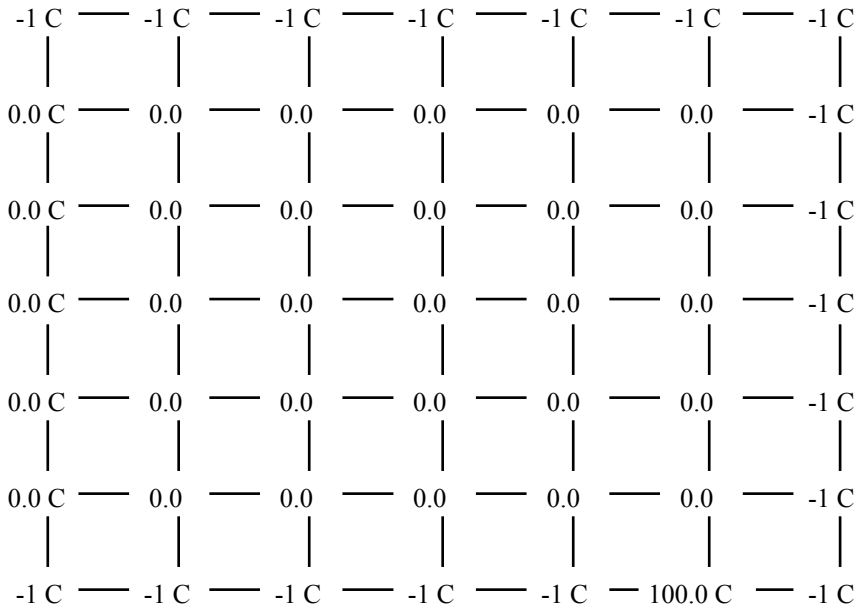

Figure 5: Implementation of our Thermal FEM on a SIMD machine

At each time step, each PE in the SIMD machine will execute the same instructions simultaneously. First, if it carries the 'C' flag, then it is marked as inactive and does no further action. Then, each active PE asks of each of its neighbors what their value is. This communication is simultaneous, with every element requesting the value of its northern neighbor in lockstep, then its western, and so forth. After gathering all this, then each processing element

changes its own value to be the average of its neighbors. After ten such time steps, the status of our model can be shown by Figure 6, below.

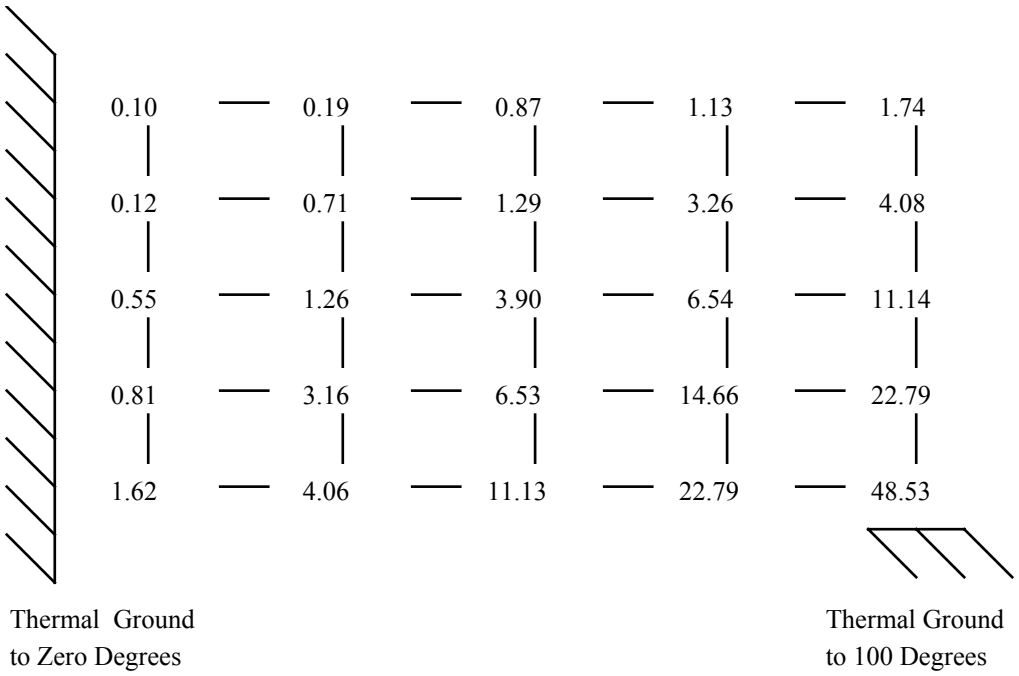

Figure 6: Condition of Thermal FEM Model after 10 Steps

A SIMD machine can do these calculations very quickly due to the rigid definition of the actions of each processing element. As our modeled plate is heated on its corner, this temperature is transmitted throughout the nodes. Figures 7 and 8 show out model at 50 steps and 100 steps respectively. Notice that we have begun to come close to a static state solution, with increasingly small incremental changes in the temperature of individual nodes.

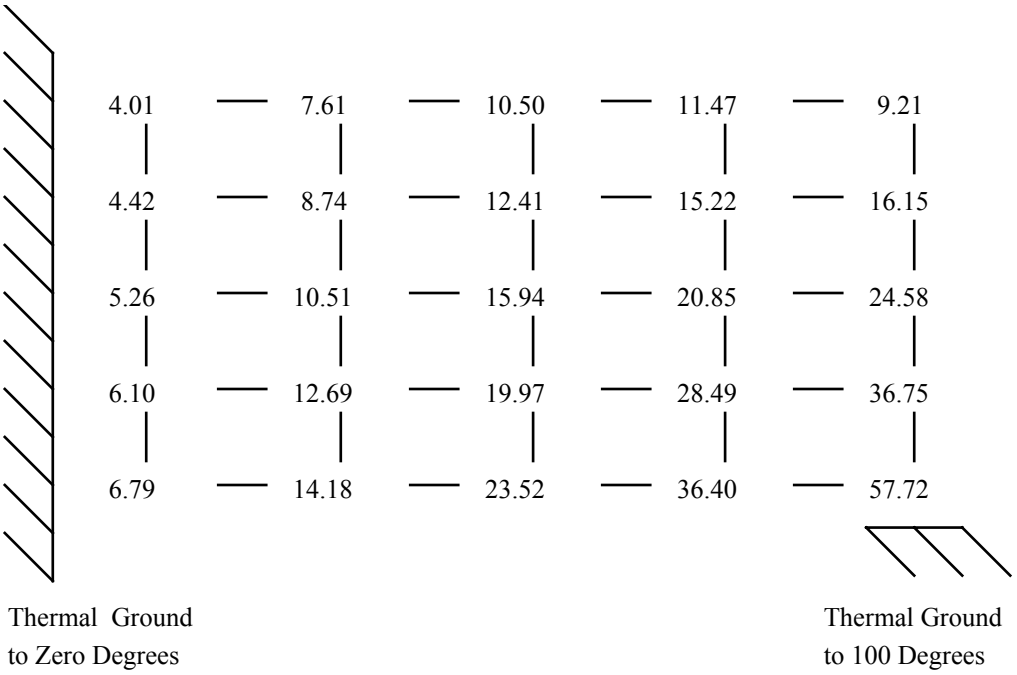

Figure 7: Condition of Thermal FEM Model after 50 Steps

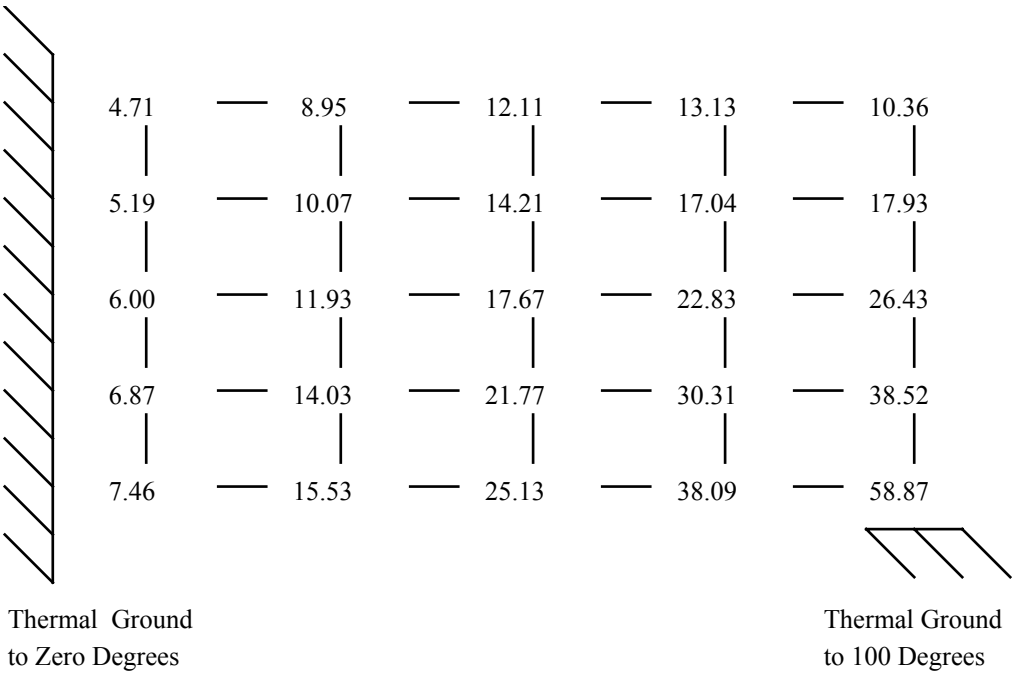

Figure 8: Condition of Thermal FEM Model after 100 Steps

This example has shown a simple problem that is easily mapped to a sample machine. However, we are by no means guaranteed that the FEM mesh representing our problem will have the same topology as our machine. Consider the example of a cantilever beam that was discussed earlier. The FEM meshes of our target problems will be two- or three-dimensional and irregular in shape, while the array of PEs in a SIMD machine such as the MasPar is best represented as a two-dimensional surface of a wrapped toroid. We are faced by the problem of the best way in which to map the nodes of the FEM mesh to the PEs of the SIMD machine. Each PE can be assigned multiple FEM nodes, within the limits of the local memory at that PE[5]. Obviously, the communications costs could be minimized by having all of the FEM nodes assigned to single PE, at which point we regress to the single-processor computer model. Thus, in order to fully utilize the power of such a SIMD multiprocessor machine we must also seek to minimize the load on each PE. A tight mapping that provides low communication costs and evenly distributes the processing load could greatly reduce total computation time.

1.3: The Genetic Algorithm Approach

This mapping problem is not a simple one, with these two heuristics of minimized communication and maximized distribution in conflict. This mapping is essentially an extension of graph theory, and specifically the graph partitioning problem. Unfortunately, this problem, and our mapping problem, is NP-complete.[6,7] We need to find a method by which we can resolve these competing constraints. In this work, we draw from an algorithm which has demonstrated its ability to produce solutions in highly complex environments of constraints: evolution. Evolutionary computation defines a class of algorithms, with genetic algorithms being the particular method we are working with[8]. Genetic algorithms take a large pool of potential solutions, in our case representations of mappings of FEM nodes to PEs, and breed from them progressively better solutions. Each solution to our mapping problem is subjected to a test of how well it fulfills our need for distribution and communication, and how well it fares in this test

is termed its "fitness". Solutions with high fitness ratings are preserved and recombined with other successful solutions. This gives us an ability to act as a preprocessor to the actual FEM problem in which we can, at any time in the process, retrieve the most fit solution to date.

Let us present a cursory example of a genetic algorithm, the details of which we will discuss in Section 3.2. This example, the SimpleGA program[9], has been created to approximate solutions to quadratic equations. We envision a possible solution to a quadratic equation being represented as 5 binary values. The first bit shows sign, and the final four bits are a big-endian binary representation of an integer value. Thus a potential solution might be shown by the string "10110", or the value -6.

For any representation of a solution to a genetic algorithm, we need to be able to formulate an objective function. In this case, we wish to minimize the equation $Ax^2 + Bx + C$, ideally finding the solution to $Ax^2 + Bx + C=0$. Thus a fitness of zero is an ideal solution to the problem.

We initially create a population. In this case the population is created entirely randomly, though this is not necessary to genetic algorithms. We have made a population of 20 potential solutions. In this case, the equation we will attempt to solve is $x^2 + 3x - 54=0$.

11100	10011
11101	00110
01000	00000
11011	10010
11111	01111
11100	11110
10101	01100
11110	01101
10101	01011
10011	11101

Figure 9: SimpleGA: Initial Population

In the first generation, we find the best 10 of these solutions. These 10 will be the parents of the next generation. Note that we already have found both of the solutions to the equation. However, a genetic algorithm needs to also have a method by which to find solutions that were not within the initial population. These are called genetic operators, and the simple genetic operator that we have implemented for the SimpleGA is a crossover. The crossover operation works much in the same way that chromosomes recombine in biology. We choose two parents from our "best of" group, and a crossover point on the solution string. The new child is composed of the first parent's solution up to the crossover point, and the second parent's solution after the crossover point. For instance, if we take the last two solutions in the initial population as parents ("01011" and "11101"), and choose a crossover point of three, then the child would be "01001". We will discuss more details of the genetic algorithm later in Section 3.2.

We can thus create a new population from this initial set of strings, such as that shwon in Figure 10. We apply our objective function to each string. We then take the ten members of that population with the best fitness and set them aside. We then use these ten to create the next population, and repeat the process.

Number	Solution	Value	Fitness
0	01101	6	0
1	10010	-9	0
2	11101	-14	100
3	01100	6	0
4	10011	-9	0
5	01111	7	16
6	10101	-10	16
7	01011	5	-14
8	10101	-10	16
9	10011	-9	0

Figure 10: SimpleGA: Generation 1

By the ninth generation, the population consists entirely of solutions for which the fitness function is zero. A genetic algorithm could be written to stop upon such a condition (as this one was) or it could continue to attempt to find more solutions.

Number	Solution	Value	Fitness
0	10010	-9	0
1	10011	-9	0
2	10011	-9	0
3	01101	6	0
4	10010	-9	0
5	01100	6	0
6	10010	-9	0
7	10010	-9	0
8	01101	6	0
9	10011	-9	0

Figure 11: SimpleGA: Generation 9

1.4: The Problem in Summation

We have applied genetic algorithms to the problem of mapping large FEM meshes to a SIMD architecture such as the MasPar series. The problems that we have tackled are related to a larger body of work currently in progress at WPI. It is desired that eventually users should be able to create a design problem in a common CAD/CAM package, and then have their problem analyzed automatically for them. This analysis consists of several steps. The geometry of the problem must be converted from a solid model form to a three-dimensional mesh. This mesh will then be mapped to the elements of a SIMD machine (such as the MasPar MP-1 used by the WPI ECE Department). The mapped mesh will then be computationally solved using a parallel FEM algorithm.[10] This work fits into the overall structure as the second step mentioned above.

Chapter 2: Background

There are three major aspects we will present in the background of this project. Thus, we will first discuss more detailed concepts of genetic algorithms, the computational paradigm used in this thesis. Then we provide some more detailed background on SIMD machines, which define aspects of the end target of our problem. A brief description of previous work in related areas will be given, which also provides a basis for evaluation of our work.

2.1: The Method of Genetic Algorithms

One way in which this mapping of FEM problem mesh to SIMD processor topology might be done is with Genetic Algorithms. Genetic Algorithms show a robust capability to drive towards optima when faced with a complex environment.[11] Genetic Algorithms are based upon biological theories of evolution in nature, and their mechanisms reflect this parentage. A number of potential solutions compete in a pool that weeds out the poorer and pushes along the better over time. By deciding how much time and resources we wish to dedicate to the Genetic Algorithm, we can choose to get a "quick and dirty" solution, or to find a more optimal solution.

2.1.1: Biological Background

The theory of genetic algorithms is based on theory from biology, namely the modern refinement of Darwin's Theory of Evolution. The characteristics of an organism are determined on the basis of the genes in its chromosomes. A complex organism commonly has many thread-like chromosomes, in the case of normal humans, 46. Each of these chromosomes has many genes in linear order along them[12]. Each gene has several different forms, called alleles. A computer scientist might benefit from an analogy to computer storage to aid in conceptualizing this. Consider memory divided into pages (chromosomes), each of which has many addressable datawords (genes). At any addressable dataword there may be a number of values (alleles).

A typical vertebrate has tens of thousands of genes in its chromosomes. This description would allow there to be about 10^{3000} possibilities, yet even a large population (say, 10 billion individuals), does not show any significant fraction of this diversity[13]. Not all of those combinations produce viable organisms that can compete in the natural world. In biology the phenomenon that keeps all the possibilities from evidencing is called epistasis. John Holland, one of the more influential fathers of modern genetic algorithms, draws an analogy:

> "The problem is like the problem of adjusting the 'height', 'vertical linearity', and 'vertical hold' controls on a television set. A 'best setting' for 'height', ignoring the settings of the other two controls, will be destroyed as soon as one attempts to better the setting of either of the other two controls. The problem is vexing enough when there are three interdependent controls, as anyone who has attempted these adjustments can testify, but it pales in comparison to the genetic case when dozens or hundreds of interdependent alleles can be involved."[14]

This combination of alleles which together yield beneficial results are called "co-adapted", and later we will refer to them as schemata. The characteristics evidenced by an organism through its genetic material is called a phenotype.

The mechanisms by which new genes are created from are called genetic operators. In biological organisms, a common operator is called "crossing-over" (or simply crossover). Given two chromosomes (in sexual reproduction, one from each parent), the chromosomes cross and break at one point, one 'header' picking up the other's 'trailer' and vise-versa. Thus two child chromosomes are created, each with genes of both parents. Another common operator is mutation, where the allele of a gene is changed due to a random variation caused by the influence of chemicals, radiation, or the like.

Children which are well adapted to survive in their environment have a better chance to go on and produce more offspring than their less well adapted siblings. Thus incremental changes occur throughout the population which yield more fit individuals. Individuals introduced into an environment have their fitness tested for survival and propagation in that environment. Being a well adapted phenotype does not guarantee that you will not meet an

untimely death before you can reproduce. The pressure of competition and the environmental problem will cause a shift towards more optimal solutions.

Thus, in summary, algorithms based on genetics and natural evolution function as follows. Working from a pool of possible solutions (or genes), the most optimal (best fitness) of these solutions are chosen to create the next generation pool of genes. These new organisms (child solutions) may be created through a number of operators such as the crossover of the chromosomes of successful parents or random variation (mutation).

2.1.2: General Genetic Algorithm Theory

Modern genetic algorithm research was pioneered by John Holland, and is described in Adaptation in Natural and Artificial Systems [1975]. In this work Holland provides a mathematical model for the process of adaptation of a generalized organism through evolutionary mechanisms. This model introduces a method of representing the genetic material or chromosome, and an abstraction of logical groups of these chromosomes: schemata. Holland's schema theory is the basis for the genetic algorithms we will consider in this work.[15]

Consider that there are a number of structures, that can define potential solutions to a problem, and call this the set A$ = {A_1, A_2, A_3, ...}. For these structures, we have a finite number of detectors which we can use to distinguish the members of the set A$. Let us call these detectors ∂_1 = {∂_1, ∂_2, ∂_3, ... ∂_1}. Each of these detectors can have an output of the set Vi = {v1, v2, v3, ... vi}.

Consider also that we have a set of genetic operators Ω = {w1, w2, w3, ... } that can be used to change state from one structure in A$ to another. For these operators, we have a plan by which to select operators to apply to our current state (T$), and an objective function by which to compare structures (X$).

In the term of genetic algorithms, these structures (A$) are the chromosomes. The chromosomes are represented by a sequence of genes (∂) which lie at given positions along the

chromosome. The alleles are the values that these genes can take (V). We must have some methods by which to change the state of our chromosome (Ω). These are our genetic operators, and the genetic algorithm determines when each genetic operator is used. We also need a selection method by which to choose our structures (T$) to operate on. We need to be able to evaluate the optimality of a given structure with an objective function, and from that derive a fitness function to compare one structure with another (X$). Given these definitions, Holland shows that genetic algorithms will converge upon an optimal solution.[16]

To create a genetic algorithm, we must decide on a representation of a potential solution, which will be our chromosome. Consider then, a chromosome in our structure space. It is represented by a string of gene-allele pairs, where the alleles might be defined to be of the set {0,1} so as to show the presence or absence of a certain feature. Most genetic algorithm work only uses one chromosome to represent an individual organism, and many use binary allele values for ease of representation. A chromosome with three genes might be expressed then as (0 1 0). We can evaluate this chromosome with our fitness function, and arrive at a value of its optimality. The problem is thus, how can we manipulate this chromosome so that it retains good qualities while shedding less favorable qualities.

Consider a set of chromosomes whose first gene has the value of 0, and whose last gene has a value of 0. We don't care what the value is of the second gene. We can express this as (0 * 0) where the asterisk is a wildcard or "don't care". This is a schema, a set of solution structures which share common features to attain greater cooperative fitness. The major point of Holland's work is that a large number of these schemata are created with each individual chromosome. Each chromosome, once evaluated, represents a data point for each of these schemata. Since we then take effort to preserve schemata in our genetic operators, Holland describes the GA algorithm is "intrinsically parallel"[17]. A large number of these schemata exist within a chromosome population, and each schemata is constantly being tested and adapted simultaneously. This parallelism of adaptation supplies the argument for why genetic algorithms

offer benefits beyond that of simple probabilistic or hill-climbing searches alone. Such searches are only attempting to prove or disprove one particular solution at a time, and thus do not have the breadth of search that genetic algorithms possess.

An important point to consider of genetic algorithms is that they are deterministic. Individual decisions within the algorithm are probabilistic, such as the selection of parents for a new child chromosome. However by using known genetic operators and selection methods, the overall action of the genetic algorithm is deterministic[18].

To evaluate these organisms (or potential solutions), we must determine the objective that we wish our solutions to converge on. The expression of this is called the objective function. Our objective function, for instance, might be to minimize the value of a multi-variable equation. These are our detectors (∂). This objective function may produce a result which is then modified by another function to create a number manipulated directly by the genetic algorithm, the fitness function. For instance, our fitness function will normalize the above mentioned result to a real value in the range from 0 to 1, with a rating of 1.00 being an optimally minimized solution.

Given a population of solutions (from A$), and a rated fitness, we now choose an operator to use to modify the solution. For instance, the previously mentioned crossover evident in biology might be used to split and recombine two chromosomes. Which individuals are chosen to reproduce in this manner is determined probabilistically by their fitness values. There are a number of these selection methods used in genetic algorithm research, as well as a number of genetic operators. Many of these methods and operators have their proponents in the GA community, and which are appropriate for a given problem is a matter of debate. These operators must be shown to maintain the deterministic nature of the genetic algorithm.

The genetic algorithms forwarded by Holland in 1975, encompass a strategy for solution of complex problems. Holland's schemata theory mathematically shows that GAs converge on their optimal solution. The genetic algorithm researcher must make several decisions, namely to

decide on a representation, an objective function, a fitness function, a selection method, and genetic operators.

2.1.3: Messy Genetic Algorithms

Messy Genetic Algorithms (mGAs) are an extension of Holland's concepts, and differ from the usual GA approach in three major ways. Firstly, they use variable-length chromosomes that may have overspecified or underspecified values. Secondly, they use a costly initialization procedure to secure all the useful building blocks needed in a problem into the population. This initialization causes the algorithm to be broken into two stages termed the primordial stage and the juxtapositional stage. Thirdly, they attack the problem over several "eras" of populations, each of which has a separately initialized population. This initial discussion of mGAs is drawn from "Don't Worry, be Messy" [1991] and "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale" [1990] by Goldberg, Deb, and Korb.

A messy genetic algorithm uses individuals with variable length chromosomes. Consider a chromosome that has 3 genes with binary values. A standard GA chromosome might express this as (1 0 1), with the unstated assumption that the first value corresponds to the first gene, etc. A fully-specified GA might express this chromosome as ((1 1) (2 0) (3 1)). This chromosome describes the alleles at each of the three gene sites, and gives one and one value for each site. However, for a messy genetic algorithm, the chromosomes ((1 1) (2 0) (3 1) (1 0)), ((1 1) (2 1)), and even () are also valid. The first is considered "overspecified", since it has two possible alleles given for gene number 1. The second is considered "underspecified" since it contains no allele for gene number 3, with the third example being a rather more dramatic example of underspecification. Both overspecification and underspecification must be handled by the messy genetic algorithm. Overspecification is handled simply by first-come-first-serve left-to-right processing of the allele-gene pairings. In other words, repetitious attempts to assign an allele to a gene beyond the first found are ignored. Note however that this "ignored" information continues

to reside in the chromosome and may be passed on to children[19]. Underspecification is a more devious problem covered by the existence of a competitive template. Any unspecified alleles are drawn from the template to create a full chromosome that can be evaluated by the objective function. The creation and modification of this template will be discussed later.

Messy genetic algorithms claim more robustness than general genetic algorithms in part through the use of an expensive primordial stage. The primordial stage guarantees the production of the optimal building blocks of a given order (k). In other words, since we create all the chromosomes that differ by only k genes, we know that the best schemata of length k exists in our solution pool. If our problem is defined by a chromosome of length l, we work in each era on a succession of current working strings of size (l-k). Thus the primordial phase needs to create a population of size n, where $n = 2^k \left| \begin{matrix} l \\ k \end{matrix} \right.$.

A major part of the appeal of mGAs is that they are supposed to be "noise-free". This means that each competitive template is guaranteed, as shown by Goldberg et al[20], to be optimal to building block order k. This is because we've already guaranteed that the best building block of length k has been created, and we only create new children by tournament selection, a one-on-one deterministic fitness trial. Since each subsequent era of the evaluation uses the previous era's winning solution as the new competitive template, you are given a ready answer that continues to approach the global optima. This also means that the mGA can be halted, and the best chromosome is always available, optimal to a specific order of building block. The reason that Goldberg and his colleagues are enthusiastic about this multi-era system is that it can defeat "deception" of a given order less that k. Consider a problem whose solution looks like a saddle function. It has two major peaks where the optimal answers might be found, and a hill-climbing algorithm or a genetic algorithm, might be misled by the lower peak having a higher slope. This is a deception of order 1. Messy genetic algorithms defeat this deception on the second era. This predictable optimality is a major selling point for Goldberg in the use of messy genetic algorithms.

The next stage is the juxtapositional stage. This stage uses two genetic operators (cut and splice) to create the competitive template for that order. This is another argument for generality in mGAs, since general genetic algorithms have been found by some to require a large amount of tuning of the genetic operators to provide good results. mGAs avoid this problem by not dealing with the plethora of available operators, mutation, or variable population sizing. Instead, there are only two operators in mGAs: cut and splice. Mutation is not used in the mGA projects described by Goldberg[21], and was not used in our implementation.

Further details on the implementation of a messy genetic algorithm is given in section 3.3.1 of this paper. However, to understand the needs of our implementation, we must also consider the hardware target that we are attempting to map to.

2.2: SIMD Machines and the MasPar

In order to be able to define an objective function for our Genetic Algorithm comparisons, we must be able to find a measure of how well our mappings will actually work in solving the FEM problems. There are several factors that contribute to the effective speed of a program running on a SIMD machine, and thus a further description of the SIMD paradigm is necessary at this point. SIMD is an acronym which stands for "Single Instruction, Multiple Data" in Flynn's taxonomy[22]. A single program instruction is carried out simultaneously by multiple processing elements on different elements of the data set. All the processing elements in the active set do the exact same instruction at the exact same time. SIMD machines are generally produced with a large number of relatively simple processing elements (PEs), each of which are loaded separately. The processing elements of the SIMD machine can communicate their neighbors during processing, at a defined constant cost. So we have three major factors influencing the speed of code execution on a SIMD machine, in our case, the MasPar. While we discuss here constructs that are specific to the MasPar, the same or similar constructs would affect implementation on other SIMD machines.

Firstly, we consider the complications of the synchronization inherent in a SIMD machine. Synchronization plays a large part in the efficiency of code written for SIMD machines. All the processors operate in lockstep synchronization on the same instruction. For instance, if the code included an If-Then-Else construct, then the processor's local data might cause them to need to execute either the Then block or the Else block. As the program executes, all the processors using the Then block would become part of the active set, and those not using the Then block would be excluded from the active set. All the processors in the active set then execute the Then block of code, while the other processors are idle. The roles are then reversed as the Else block of code is executed. Thus, the way in which the code is written for a SIMD machine is critical. For our purposes, we can largely ignore this primary source of SIMD problems, as we wish to make the mapping independent of solver implementation.[23] However, the nature of synchronization will show up in other considerations.

Secondly, we consider the raw loading of the processors. The most obvious measure of speed on a multi-processor machine is the loading of the tasks. The potential computational penalties for loads of varied weights on individual PEs is even more critical on a SIMD machine than on other kinds of multiprocessing paradigms due to synchronization. If one worst-case processor has to loop and do n jobs, then all the processors will spend the same amount of time either taking care of their jobs or remaining idle outside the active set waiting for the worst-case processor to finish. As an additional complication, the individual PEs have limited local memory. If the problem assigned to the PE is larger than the local memory, then it will need to retrieve the remainder from the controlling machine when that information is accessed. The entire machine will then go idle while the communication with the controller[24] takes place. Thus, there are a number of issues in loading the PEs with problem elements from the FEM mesh. The basic thrust is that we wish to have an evenly distributed load across the PEs.

Thirdly, we consider the communication costs of solving the mapped problem. There are two constructs which can be used for the communications between PEs in the MasPar, the Xnet

and the router. The Xnet is a high speed hardware communication system which connects each PE with its eight neighbors. The MasPar-1 is laid out in a grid of 1024 elements, 32 rows by 32 columns. The Xnet wraps around this grid vertically and horizontally, which makes the MasPar more properly considered a wrapped toroidal topology. Figure 12 shows the hardware connectivity as if the MasPar were only five by five rather than 32 by 32 in size. Note that this connectivity does not change if only a smaller subset of the nodes is active. The router construct allows communication between any two elements in the MasPar. The router is slower than the Xnet, and as mentioned before, if one PE needs to make a call to the router, then all PEs will go idle waiting for that to be completed.

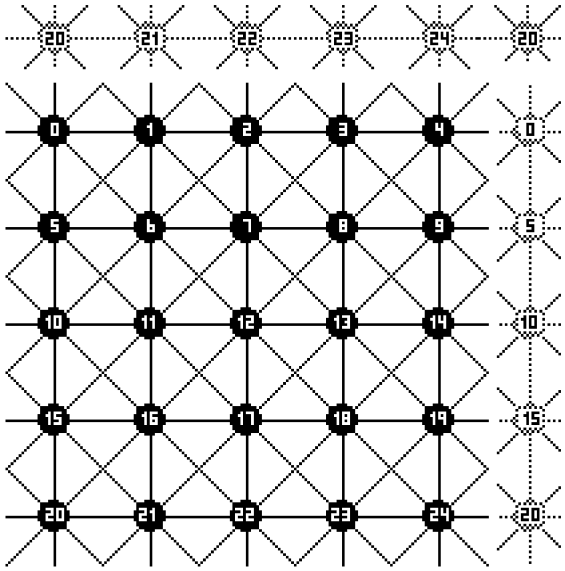

Figure 12: Miniature MasPar Xnet Connectivity
(grayed elements show wrap-around)

Consider the FEM Mesh of a cantilever beam that was illustrated earlier. If we take that mesh, and number the nodes for easier recognition, then we get Figure 13, below. Our problem contains an aspect that provides a simple challenge to mapping, namely that it is wider than our

example miniature MasPar. Thus we cannot simply place the problem down upon the surface of the MasPar, but must make room for the nodes at the end of the cantilever.

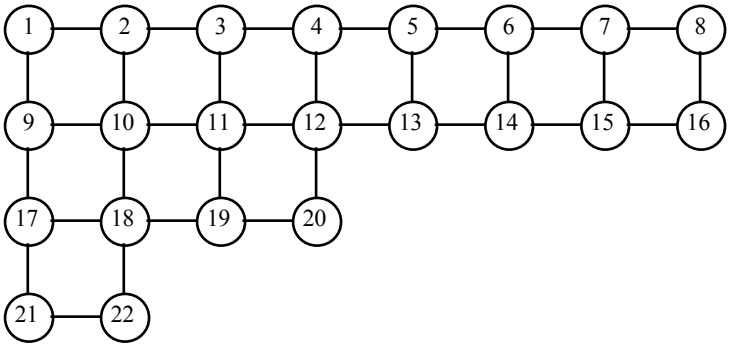

Figure 13: Cantilever Problem with numbered Elements

One way that a mapping might be created is shown in Figure 14 below. In this mapping, we mange to utilize 88% of our processors, without overloading any of them. Our communication costs is only slightly higher, since node 13 needs to make two jumps to reach node 14. Node 5 exploits the wrapped toroidal geometry, and reaches Node 6 in one step. However, the fact that we have to wait for nodes 13 and 14 to communicate means that every processor will go idle for the time it takes for the MasPar to make one Xnet communication.

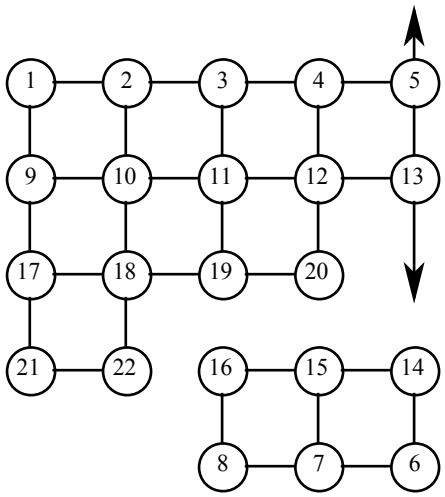

Figure 14: A simple mapping of FEM Mesh to miniature MasPar PEs

This is, of course, a very simple example of a mapping problem. A real world problem might well have a three dimensional mesh, which presents similar problem, but in much greater complexity. These larger problems quickly escalated beyond what it is desirable to do by hand. It is to handle this complexity that we turn to other algorithms, including genetic algorithms.

2.3: Related Historical Work

One related work is that of Storøy and Sørevik [1992], who describe a solution to a similar problem, the Dense Linear Assignment Problem (LAP). The LAP maps members of a connectivity graph onto a cost-metric graph in such a way that the cost is minimized. They implemented two different schemes of the Hungarian algorithm for solving the LAP, primarily using a MasPar MP-1. While they report that their results are arguably optimal, the time required is bounded by $O(n^2)$ when using this parallel algorithm, which is reduced from $O(n^3)$ in the serial algorithm.

Bokhari [1981] tackled the problem of mapping FEM nodes to the processors of a parallel machine. He used a Monte Carlo search method that initialized with a random allocation of processors to problem elements. The algorithm then modified the current working mesh by swapping one node-PE pair in the way that most increased his fitness function, performing an exhaustive search of all such swaps. When his algorithm reached the top of the hill thus climbed, it checked to see whether this was better or equal to the last saved result. If it was, it saved this new result and took a random jump to another space. Bokhari's algorithm is discussed in more depth in Section 3.4. As we will discuss later, his original algorithm does not fully handle the problem as we desire. However, a modified form of this algorithm was used as our test case which we could compare our genetic algorithm work to.

Chapter 3: Approach

The genetic algorithm is applicable to our problem, as we will show. We will then describe how the details of the genetic algorithm and messy genetic algorithm are implemented. We will also describe our implementation of Bokhari's algorithm, which will be used for comparative purposes.

3.1: Problem Applicability

One important qualification must be made concerning the algorithms we used for mappings. The original Bokhari algorithm called for mappings of at most one problem node to a processing node. In our first exploration of the genetic algorithm approach, we retained this limitation. This, however, limits the problem size to the size of our parallel machine. Seeing as we wish to find an algorithm capable of mapping very large problems, this needed to be modified. In the following discussions, thus our primary data comes from the multiple-mapping modification of the Bokhari and GA approaches.

Our general problem can be applied to the GA approach in the following manner. An evolutionary organism with one chromosome consists of a number of genes equal to the number of elements in the problem mesh. Each of these genes can take on a value corresponding to any one of the processing elements of the SIMD machine. Thus we can represent any mapping of problem elements to SIMD processing elements.

3.2: Genetic Algorithm Approach

We have a number of parameters to work with in designing our genetic algorithm for a given task. We must determine what our objective function is, and how our fitness function expresses this. Given this, we choose a selection method to produce the candidates for parenthood of the next generation. If there are any controls on the dynamics of the population, this may place an additional constraint on the production of new candidates. Next, we decide

what genetic operators are available to create this generation. Finally, we implement the details of how large each population is, and for how many generations we will seek our answer.

3.2.1: Objective and Fitness Functions

While it is necessary to have some knowledge of how the MasPar works to make an appropriate objective function, the key word is appropriate. We realized immediately that the objective function would not, for instance, return the time necessary to solve the target mapped mesh. We are not interested in taking the solver code and developing a profile specifically for it. Considering that this objective function must be run on every potential solution, it must not be, an expensive computation. If our object function takes an excessive amount of time to compute, the advantage gained by creating the mapping will be lost in the resources necessary to create the mapping in the first place. Instead, we wish to find a good estimatator for code which has not even been written yet by considering the overall problem on a more abstract level.

As an example of a basic objective function, Bokhari [1981] used a comparison of edges. If there was an edge in the graph of the problem mesh, there should be an edge in the graph of the processor mapping. His fitness function was the maximization of the cardinality of these correlating edges. But this only measures full "hits" on the solution. For instance, it does not consider that replacing a problem edge with two processor graph edges is better than five processor graph edges. As we have mentioned, if 69 out of 70 edges correlate correctly, but the seventieth edge requires 10 Xnet communications, then all the processors will wait while the slow edge is being dealt with. Thus it would not be better than if all of the processors had to do five communications on the Xnet, reliably and consistently.

Bokhari also allowed a single problem element to be mapped to any single processor element. Since this would be required to do the kind of problems we are targeting, this also adds the issue of processor loading. What we wish to avoid is overloading one processor while leaving others to idle.

We do not necessarily seek an optimal solution. We only have an estimator of the final performance of the solver, and so to spend a large amount of processor time optimizing the estimator may not pay off in great gains in performance. Instead, we wish to act as a pre-processor for the problem which will result in a reduction of the overall time necessary to solve the FEM mesh. So the computational time required in making the mapping should not exceed the gains in actual solving time (compared to an unintelligent) mapping.

Our concerns were handled by our final objective function, which computed the sum of the variance of the communication costs and the variance of the processor load across all of the problem nodes. Thus we seek the minimization of this objective function. To create a fitness function, we inverted the objective function and normalized it over the range {0,1}. Thus a fitness rating of 0 indicates the absolute worst case possible for both processor loading and communication costs, while 1 indicates that all of the processor elements have the exact same loading and the communication costs are all equal. Note that it is quite likely that the optimal possible solution is less than 1.

3.2.2: Selection Methods and Population Dynamics

Our GA uses roulette wheel selection, which means that the chance of any given individual being chosen is proportional to its fitness. This selection method has been shown to work within the constraints of Holland's theory[25]. Consider a roulette wheel in which the slots are not of equal size. Each of the slots is of a size corresponding to its fitness. Thus the ball might fall in the narrow slot of a unfit solution, but more likely to fall in the wider slot of a solution judged to have a higher fitness. In our research, we found that it was useful to actually exaggerate a given organism's advantage by making the roulette wheel slot the size of the square of the fitness. The downside of this is a loss of diversity, as it makes it easier for the slightly more fit solutions (which may be identical) to dominate the wheel.

Our GA uses generational reproduction, which means that all the individuals in a population are replaced at once.[26] We do not use elitism, a method which allows a fit solution to move on to the next generation unchanged, potentially providing immortality. We also do not forbid duplication of a solution in the solution pool, potentially allowing clones to take over the pool. Both of these are factors that can influence the time to solution, but we leave them out of this work primarily to control the already large number of variables inherent in this research.

3.2.3: Genetic Operators

The GA code that we have produced is setup to easily switch between two kinds of crossover operators: partially matched cross-over (PMX) and order crossover (OX). These are the two primary kinds of genetic operators identified by Goldberg [1989] as falling within Holland's schemata theory. We used mutation in our final version of the GA, with there being a one in one hundred chance of a chromosome having one gene's allele changed.

Partially matched crossover was first used in the blind traveling salesman problem. In the traveling salesman class of problems, the primary concern is the order in which individual problem nodes (in this case, cities) are traveled. Also, no node can be traveled through more than once. This is similar to the original single-mapping approach that we used to tackle the mapping problem, where each processing element could only be assigned one problem element. Simple two-point crossover would violate that condition.

In the PMX method, two points are chosen along the length of the chromosome. Both of the parents are divided at these points, and interchange the middle segment thus cut. The pairs in the exchanged segment, however, now may be repeated more than once in the new child chromosomes. To rectify this, the original nodes which are being repeated are switched with their opposite mate. For instance, we have two parent chromosome strings with two division points chosen, A and B. Within the chosen cut string, the 1 in A and 4 in B and exchange, along

with the 9 and 7. However, this leaves repetition in each chromosome, so in the uncut string the 4 in A is switched with the 1 in B, and the 7 with the 9, leaving:

A:	3 7 4 2 | 1 9 | 6 5 8	A':	3 9 1 2 | 4 7 | 6 5 8
B:	1 8 9 3 | 4 7 | 6 2 5	B':	4 8 7 3 | 1 9 | 6 2 5

This tends to keep the general order of the chromosome from being disrupted, while maintaining the number of references to a given node.

By contrast, the order crossover works somewhat differently upon the chromosome. Like PMX, the order crossover chooses two division points and thus a segment to be exchanged. However, instead of swapping the inner segment, OX removes all of the nodes that correspond to the mate's inner segment. This leaves "holes" in each of the chromosome. These holes are then filled by sliding the genes together from the second selection point and moving the "holes" to the cut segment. Then the cut values are exchanged as in the partially matched crossover. For instance, consider our previous example chromosomes. In chromosome A the 4 and 7 are replaced by holes, as are the 1 and 9 in B. Then the uncut strings are slid together, and the cut segments exchanged, as shown by A' and B', below.

A:	3 7 4 2 | 1 9 | 6 5 8	A':	5 8 3 2 | 4 7 | 1 9 6
B:	1 8 9 3 | 4 7 | 6 2 5	B':	2 5 8 3 | 1 9 | 4 7 6

This tends to maintain relative ordering of one node to another, rather that the absolute position maintained by partially matched crossover. However, for our purposes we expected that PMX would produce more constructive juxtaposition of mapping sequences. Small problem testing appeared to back up this decision, so we proceeded.[27]

Mutation is currently considered only a minor actor in biological evolution of sexually reproducing organisms. In general for our purposes, it provides a disruptive effect and adds some measure of diversity to the population.

3.2.4: Sizing of Populations and Length of Run

There currently is much argument in the GA community about the sizing of populations. As might be expected, there is a critical tradeoff between time to solution and the quality of the solution. Small populations are more quickly dominated by above-average fitness schemata. This could potentially shut out schemata that, if given time to co-adapt fully, could produce better results. The larger populations have more room for these locally suboptimal but globally more optimal schemata to develop and recombine. The larger the population, the more computational work that must be done to complete each generation, but the more diversity is maintained. Goldberg has produced work that encourages smaller population sizes.[28]

The flip side of the problem is how long a GA trial is allowed to run. In this work we have tried a variety of population sizes and run lengths to see the effects on the result. We emphasize that in our particular problem we are acting as a preprocessor to another computational processor, the actual FEM solver. We envision that the final version of such a preprocessor would have a more dynamic halting condition, where computational time expended would be balanced against the slowing gain in solution quality. As we have mentioned in our problem definition, the mapping problem will be driven to a quick and dirty solution rather than concern in finding the elusive global optima.

3.3: Messy Genetic Algorithm Approach

One of the original intentions of the messy genetic algorithms is that it should have a minimum of parameters that need be determined by the researcher.[29] However, there still are a number of decisions that need to be made. We first take a more detailed look at the scheduling of an mGA era, and determine the open parameters there. We then look at the other variables required for the mGA, particularly concerning the genetic operators. Finally, we show the technical limits which placed a real-world boundary on our mGA work.

3.3.1: Scheduling a Messy Genetic Algorithm Era

As mentioned earlier in Section 2.1.3, the mGA process passes through two phases during an era, the primordial phase and the juxtapositional phase. At the beginning of the juxtapositional phase, the population consists of all the possible building blocks of the order appropriate to the era. This is potentially a large number (see Section 3.3.3), and the population needs to be trimmed down before it reaches the juxtapositional phase. Figure 15, below, illustrates the control flow of the messy genetic algorithm.

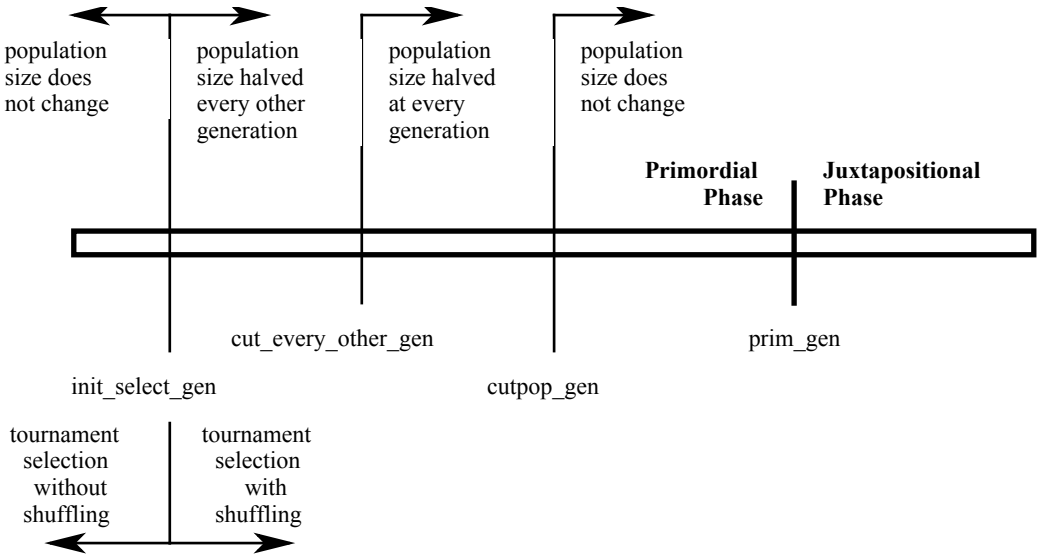

Figure 15: Messy Genetic Algorithm Scheduling[30]

Initially, the population does not change, and neighboring chromosomes are tested against each other with the winner taking both slots in the population. Then this selection method (called tournament selection) is modified by randomly shuffling the participants in the tournament. At this time, the population is cut in size every other generation, with only the winners of the tournament selections surviving to the next generation. Note that at this point, no genetic operators are being used to modify the building blocks. Then the population pressure changes again, with only the tournament winners proceeding at every generation. Then, at the end of the primordial phase, the population size is held constant again.

The placement of the signal events (init_select_gen, cut_every_other_gen, cutpop_gen, and prim_gen) determine the population pressure placed on building blocks, and the final population size that must be carried through the juxtapositional phase.

After brief experimentation, we used a constant set of these values for all of our runs. Tournament shuffling and population reduction was begun by the init_select_gen flag after one generation. The reduction of population at every other generation was ended (and accelerated to every generation) at the cut_every_other_gen flag on the third generation. After six generations, the population size was stabilized by the cut_pop_gen flag. The primordial phase ended after the 9th generation, and the juxtapositional stage then ran until the 50th generation.

3.3.2: Other Messy Genetic Algorithm Variables

During the juxtapositional phase, the population size does not change. Now the genetic operators come into play, recombining the culled building blocks. As mentioned before, there are only two genetic operators in the mGA algorithm, cut and splice. Cut takes place on a given chromosome with probability $p_C = (l' - 1)/l \bullet p_k$, thus becoming more likely as the chromosome string (of length l') gets longer and closer to the problem size (length l). Splice joins together the resulting strings with fixed probability p_S. For our experimentation p_k was fixed at 0.5, so that a full size chromosome would thus have about a 50/50 chance of being split. We kept p_S fixed at 1.0, which kept the population handling simpler since two parents always produced two children, a method also used by Deb and Goldberg in their "mGA in C" work.[31]

Mutation in mGAs could be applied either to the gene value or the allele value, unlike a normal GA where one can only mutate the allele. However, Goldberg does not use mutation, and neither do we in our work.

3.3.3: Technical Limits of the Messy Genetic Algorithm

Earlier we alluded to difficulties in implementing our problem using a messy genetic algorithm. Until now, our major concern about complexity was time complexity. After all, this is the general measure of choice when deciding is one algorithm is superior to another. Let us look briefly at the complexity of the mGA approach to our problem.

The complexity of the overall mGA operation is $O(l^k)$[32], which is driven entirely by the initialization phase. So further work by Goldberg, et al. was motivated in reducing the primordial phase to produce a probabilistically complete initialization (PCI) of the population[33].

The methods for PCI draws on another paper, "Genetic Algorithms, Noise, and the Sizing of Populations", by Goldberg, Deb and Clark. In this paper, mathematical evaluation of the objective function is used to create formula to determine the needed population size. Using this, Goldberg's GALab research group went on to create fast messy genetic algorithms (fmGA), which are distinguished from mGAs in that they have a much shorter primordial phase. In fact, the new primordial phase is $O(l \log l)$, just like the other phases[34].

Now, if we could draw directly on the fmGA methodology, then our prospects would be blindingly bright. However, there appear to be some points of concern with the application of mGAs and fmGAs to our problem. Firstly, the mGA predictions are based on the apparent assumption that function evaluations are done in constant time. However, the best methods by which a mesh mapping could be evaluated are done in $O(l)$ time. This modification will take our execution time to $O(l^2 \log l)$. Secondly, it appears to be assumed that the number of iterations of the whole mGA mechanism is done a constant number of times. However, in the paper text it seems to me that we need to iterate k from 1 to l. In practical application this iteration is made with steps greater than 1, which places another $O(l)$ loop around the whole mechanism. So now the expected complexity stands at $O(l^3 \log l)$. This, however, only describes the time complexity.

Space complexity of the messy Genetic Algorithm is also problematic. Consider our space complexity algorithm parameterized by three values: the number of nodes in the problem

mesh (the problem size, or *prob_size*); the number of processor elements to which the mesh will the mapped (the machine size, or *mach_size*); and the order of the building blocks which we are solving for, which is also the era of the mGA approach. The number of unique genes in our problem is thus *prob_size* • *mach_size*, and the number of unique chromosomes (building blocks) of order (*order*) is $\binom{prob\_size \bullet mach\_size}{order}$. And in the mGA methodology, we must create all unique building blocks in the setup of the primordial phase.

Now this does look bad, but consider a sample problem with a trivial two-dimensional mesh of 24 problem mesh nodes on 6x6 FEM machine. This yields (24 • 36) = 864 unique genes. So, in the first era (with order 1 building blocks) we'll generate 864 chromosomes. Once that is completed, we make a new competitive template, and run era #2 (order 2 building blocks) of 372816 chromosomes. Then we have era #3 (order 3 building blocks) with 107×10^6 chromosomes, era #4 (order 4 building blocks) with 2.3×10^{10} chromosomes, and so on.

Now when one tries to allocate memory for this space, as one might innocently try to do for an order 3 problem in such a trivial test, we find that if each chromosome requires a mere 32 byte header and 12 bytes per gene-allele pair, we need: 107×10^6 chromosomes * (32 bytes + 3(12 bytes)) = 7 Gigabytes of memory. Unfortunately, the hardware available to us (see Section 4.2) does not include seven gigabytes of swap space. And again, this is a trivial 24 node, two-dimensional problem. Thus there exists a serious difficulty in using messy genetic algorithms on reasonable sized problems with our current technology.

3.4: Bokhari's Algorithm Approach

In a 1981 paper, Shahid Bokhari introduced a method by which to solve the mapping problem. He proposed a method to map a FEM mesh to an "eight-nearest neighbor" FEM machine having communication connectivity much like the MasPar. This problem is very close to the problem that we describe, and thus, we have chosen it as a baseline by which to compare our work.

3.4.1 Description of Approach

Bokhari's algorithm begins with an adjacency matrix for the FEM machine. We then map nodes from the problem mesh to the processing elements of the FEM, first node to first element, second node to second element, etc. Given this mapping, we evaluate it for fitness. In the original algorithm, a mapping gets a fitness point for every pair of nodes which are adjacent in the problem which are mapped to adjacent processing elements. Thus, we are trying to maximize this mapping cardinality value.

The main loop of the algorithm then takes each individual problem node, and consider exchanging it with each other mapping position in turn. Thus all of the current mappings "neighbors" are explored. We select the change which leads to the greatest gain in fitness. If this gain is nonnegative, then we make the exchange. This new mapping becomes our current mapping, and we again check all the nodes for possible advantageous exchanges.

If the mapping finds that the best possible exchange gain is negative, then the exchange is not made and the current mapping is saved as the best mapping found. We then take a random jump in our mapping space, exchanging randomly n pairs of nodes, where n is the length on the side of the FEM machine. If one of these jumps leads us to new best position that is at least equal to the previous best mapping found, then we will continue to jump in search for better mappings. If a jump cannot climb to at least an equally fit mapping, then the algorithm terminates.[35]

3.4.2 Performance

Bokhari's work centered on 25 to 39 node problems, and running on a CDC Cyber 175, he had no problem getting back timely results. However, his algorithm executes in $O(n^2)$ time, which he admits in this article "will probably not be suitable for very large arrays (say 32x32). For such arrays, entirely different heuristics will need to be developed."[36] The work planned to target

such larger arrays, and in fact 32x32 is the size of the MasPar available to WPI for solving these problems.

3.4.3 Limitations and Subsequent Modifications

Bokhari's algorithm also depends on the problem being smaller than, or of equal size to, the FEM machine. As we have mentioned above, we desired a multiple mapping algorithm. Thus, the Bokhari algorithm needed to be modified. This was done by simply wrapping around the initial mapping when the problem is bigger than the target machine. Nodes are still handled singly, and thus can still be swapped in the original manner.

The fitness function used within the Bokhari method had to be changed to bring it into line with the GA and mGA. The same function was thus used in all three methods.

Note that the Bokhari method is also capable of finding itself in an infinite loop. Consider a problem space whose fitness function has a number of equally fit local solutions. Bokhari's solution could find itself in one of these mesa-like areas, and randomly jump to another such plateau. Since the new best fitness is neither better nor worse than the previous solution, the algorithm randomly jumps again... back to the original location. This pattern could be repeated, potentially infinitely, depending on the topology of the fitness function space. In our work, a "timeout" value was introduced to the Bokhari algorithm. A counter was initialized when the Bokhari algorithm began the jumping stage. This counter was incremented every time that a jump resulted in no increase in fitness. If the counter reached the timeout value, then the Bokhari algorithm terminated and reported this fitness as its best. If a better solution was found, then the counter was reset.

With these modifications, the Bokhari algorithm was comparable to the genetic algorithms in capability of handling our problem. As a more standard hill-climbing algorithm, Bokhari became our test case that we might compare our results to.

Chapter 4: Evaluation

In this chapter we examine the actual performance of our algorithms. In doing so we need metrics with which to measure performance. We explain the performance measures forwarded by De Jong, which are widely accepted among the genetic algorithm community, and which we used in this work. We note the hardware and software platform on which the tests were run. We present the test meshes that we used in testing our algorithms, and the results that were obtained.

4.1: De Jong Performance Measures

Within this paper, we refer to two measures of performance in evaluating a given trial, or run. These measures are the De Jong online performance and the De Jong offline performance, drawn from De Jong's dissertation, "An Analysis of the Behavior of a Class of Genetic Algorithm Systems"[37]. The De Jong online value is the running average of the fitness of chromosomes up to and including the current trial chromosome. The De Jong offline value is the running average of the best, or most fit, chromosome that has ever been seen at the current time. De Jong's differentiation of the two was to emphasize the difference in applications, in the offline case where the best solution can be saved for use at the end of the run. In the online case, the value is placed on the larger experimental whole of the evaluation run.[38]

In our work, both De Jong values are calculated. We do save the best chromosome in the GA, mGA, and Bokhari algorithms for our final solution, and thus the offline value is effectively the final measure of performance for our problem. However, the online value is useful in providing a measure of the ongoing performance of the total chromosome pool.

4.2: Hardware and Software Platforms of the Tests

Our tests were run on Sun SparcStation 2's in the WPI CAD Lab. These SparcStations were running the SunOS 4.1.3 flavor of the UNIX operating system. The program development was done using C++, with compilation done with GNU C++ version 2.5.8[39].

These machines were not dedicated solely to making these test runs, but were under use by the CAD Lab's students and staff. Thus, to keep performance of the CAD Lab at an acceptable level, most of these runs were run with low process scheduling priority ("nice'd") during the day, and raised to a higher priority during the night. Complete run data is given to provide further information on this condition. CPU utilization was given via a function call to the getrusage() function. In general, the more valuable of the two CPU usage values is the time spent in user mode, which excludes time spent in kernel calls such as process swapping, as well as I/O. This value is thus less affected by the presence or absence of competing jobs on the computer.

4.3: Test Meshes and their Results

A number of test cases were run against our algorithms to judge performance. The modified Bokhari algorithm (noted as Bokhari2) is used by which to compare the genetic algorithm (note as GA2) and the messy genetic algorithm (mGA). Some test cases were drawn from Bokhari [1981], and some were created to better emphasize the needs of our problem. Discussion of important elements of the results are given.

4.3.1: Tower25

The Tower25 test is a 25-node representation of a two-dimensional tower taken from Bokhari [1981]. The Tower25 test has a maximum connectivity of 8, by which we mean that any given node might be connected to as many as 8 other nodes in the problem mesh. This was mapped to a 5x5 target SIMD machine.

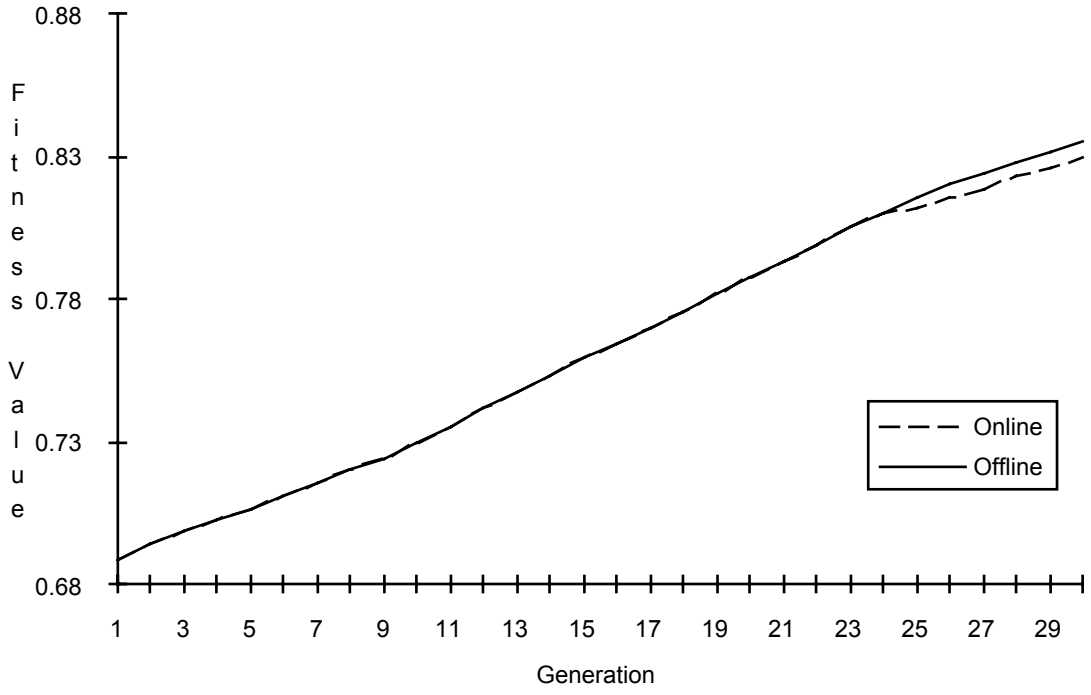

Figure 16: Performance of Bokhari2 Algorithm on Tower25

This test shows the kind of results that we expect from Bokhari. It shows a continual rise in fitness over the duration of the run. This is since by definition the Bokhari algorithm will never replace its current "best" solution with an inferior solution with the "one step away" method. However, in the end condition the Bokhari algorithm begins to "jump" to random locations in the solution space, which may be inferior to the previous generation's best solution. The beginning of this jumping can be seen where the De Jong online and offline values diverge. Soon after jumping begins, the algorithm finds that it can no longer find better results and terminates. This test ran in 80.120 seconds.

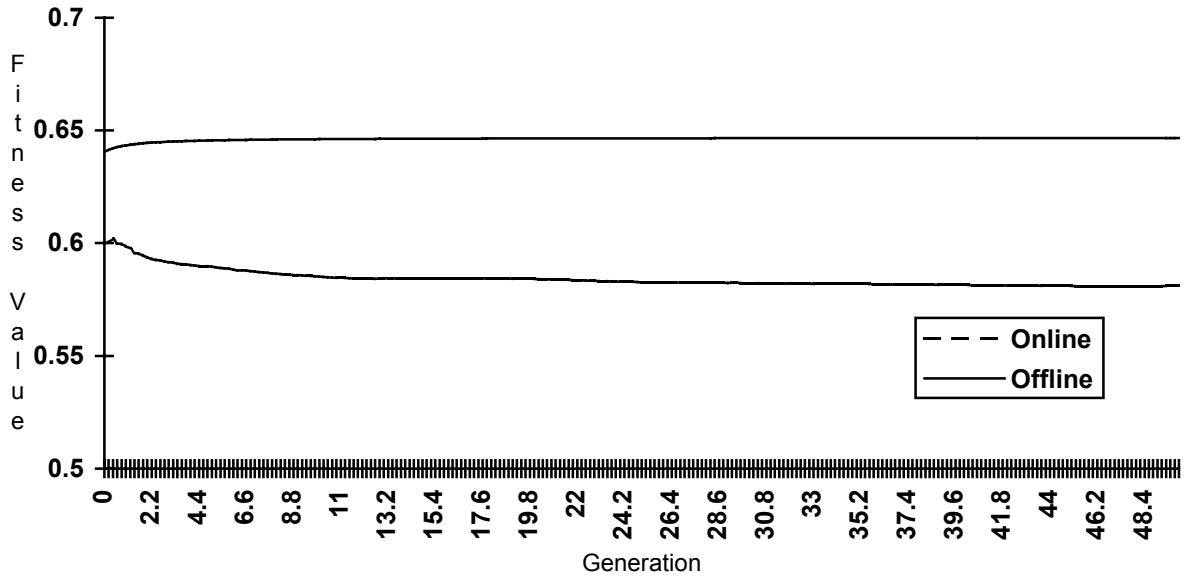

Figure 17: Performance of GA2 Algorithm on Tower25

This graph shows the genetic algorithm performance on the same two-dimensional tower that was just analyzed by the modified Bokhari's method. Distinctive differences from Bokhari are three-fold. Firstly, the GA2 has a very brief rise to a plateau and then remains flat, unlike the constant improvement of the Bokhari algorithm. Secondly, the De Jong online value begins distinctly lower than the offline value, and remains and quickly falls to a lower plateau. Thirdly, the GA performance is not only initially lower than the Bokhari, but it is unable to find a better solution. If this were because the GA was becoming trapped at a local optima, then we might expect the entire population to become full of this sub-optimal solution. This would be noticeable as the online value converges with the offline. Instead, our online and offline values show that there appears to be diversity in the population. This might indicate that any fit solutions are quickly being broken up by the genetic operators and returned to a average lack of fitness. This test ran in 25.57 seconds.

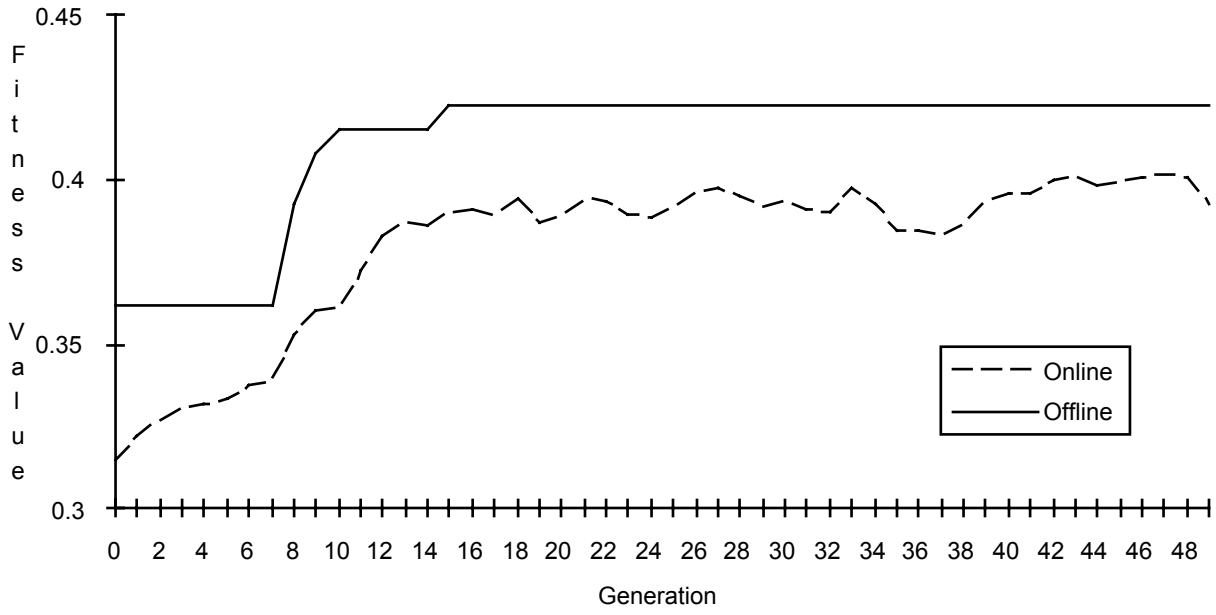

Figure 18: Performance of mGA Algorithm on Tower25

The messy genetic algorithm, or mGA, has a pattern that is different from both the Bokhari algorithm and the conventional GA. Most noticeable are the four points in time where the parameters of the algorithm change, noted in the mGA section as init_select_gen, cut_every_other_gen, cutpop_gen, and prim_gen. Once the algorithm passes the prim_gen state, it is essentially behaving as a normal genetic algorithm. As soon as this occurs, the improvement plateaus. Note that when we start the mGA, we are using building blocks of only one mapping (order one), which unsurprisingly has a very low fitness. Notice also that the online De Jong value remains in flux, probably indicating that there is a large degree of diversity in the solution pool. As is noted later, there is a possibility that if the points in time when the algorithm changed were better tuned, a better solution might have been reached. We found that longer intervals were not necessarily better (see Data), and were discouraged from long intervals because of the massive time cost of processing the early generations. Run data was not available for the mGA

runs, however it can subjectively be said to be considerably longer than the comparable Bokhari run in every case.

4.3.2: Mesh33

This test is also taken from Bokhari [1981]. Mesh33 is a roughly 3x11 latticework, with a maximum connectivity of 6. It is mapped to a 5x5 target machine. Note that unlike the Tower25 test, this means that the mapping algorithm will be forced to place multiple problem nodes on an individual machine node.

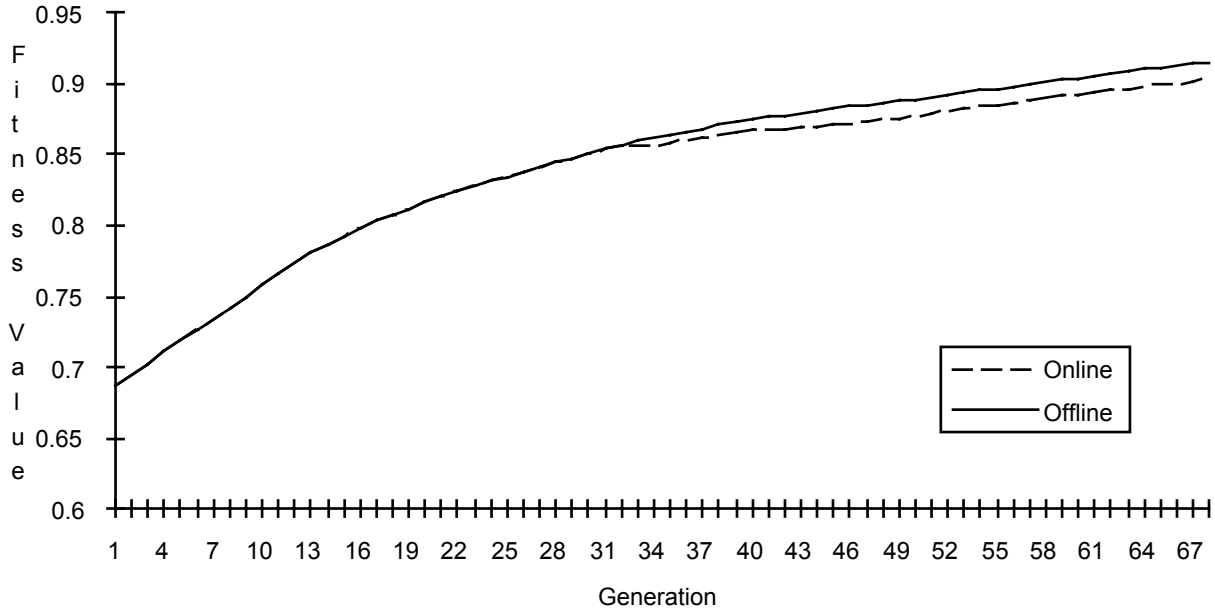

Figure 19: Performance of Bokhari2 Algorithm on Mesh33

This test shows a behavior very similar to the Tower25 test above. Note that in this case the jumping stage of the Bokhari algorithm was held for a longer period of time before it was unable to improve. This test ran in 682.93 seconds.

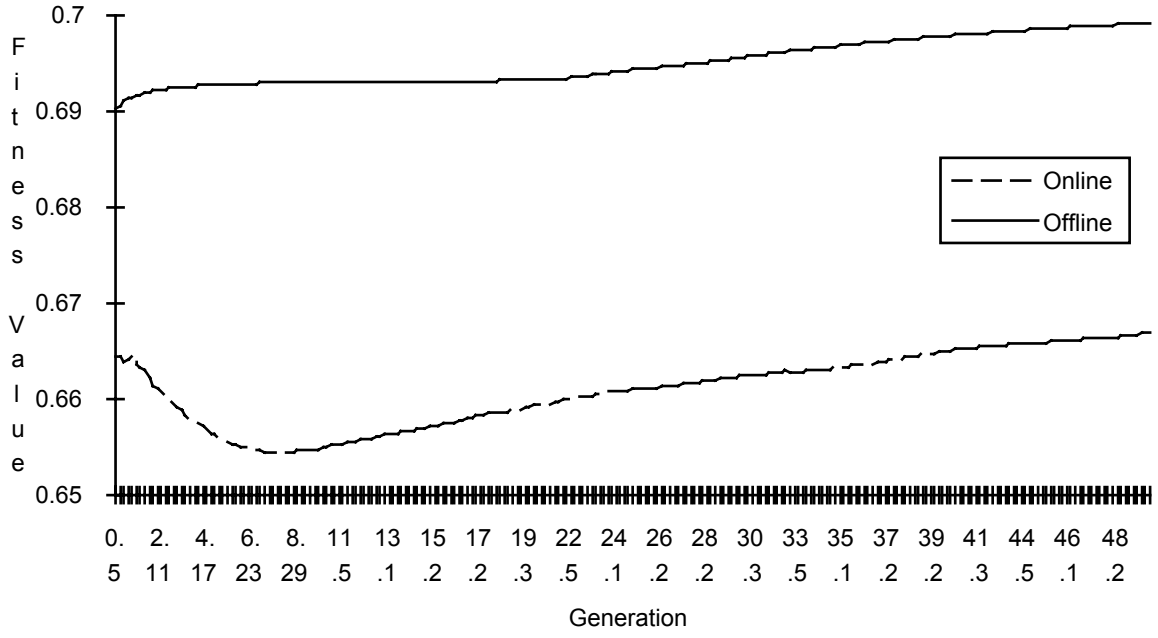

Figure 20: Performance of GA2 Algorithm on Mesh33, size 30 population

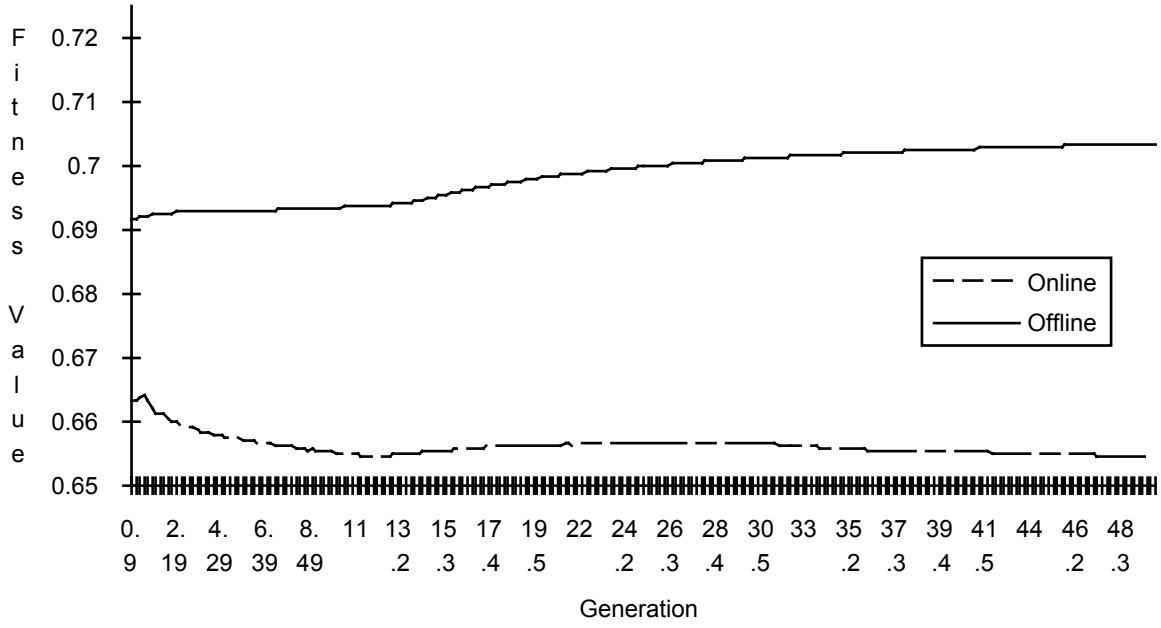

Figure 21: Performance of GA2 Algorithm on Mesh33, size 50 population

Here we present two different runs of the GA2 against the 33-node two-dimensional mesh. The difference between the two runs is an increase in population size from 30 to 50.

Again, the genetic algorithm follows a relatively flat fitness path, with the more populated run producing a solution that is only a small fraction more fit than the smaller run. The same basic behavior is seen as in the Tower25 test, with the exception that in the smaller run the online value is rising towards the end of the run. This may indicate that the population pool is becoming more homogeneous. This test ran in 32.51 seconds for the size 30 population and 59.14 seconds for the size 50 population.

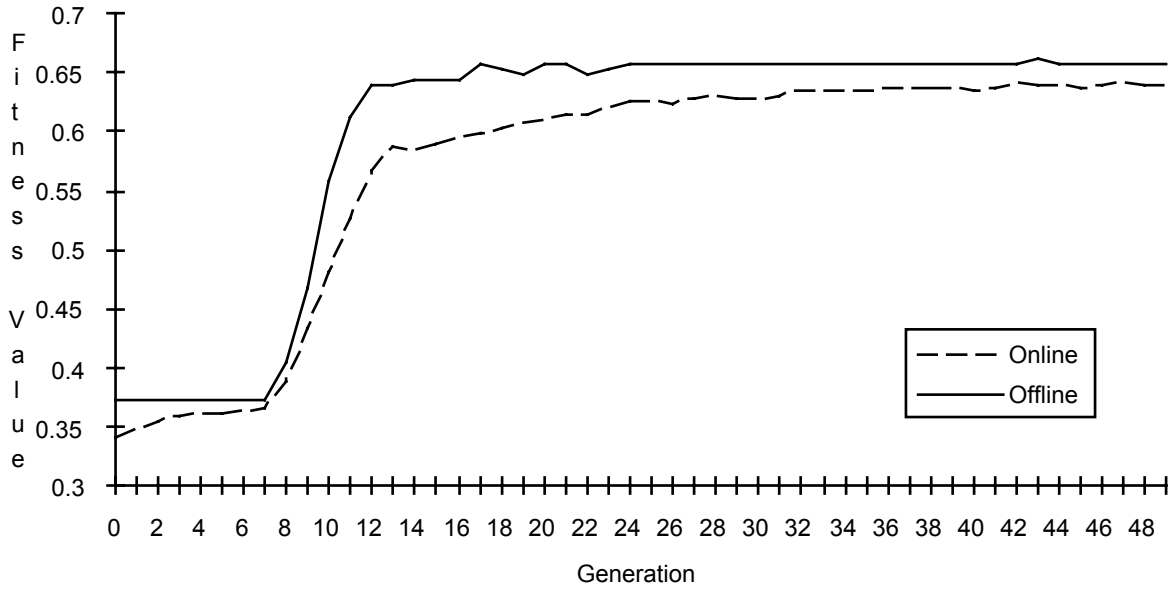

Figure 22: Performance of mGA Algorithm on Mesh33

While not as clear cut as in the Tower25 test, the messy genetic algorithm is here also exhibiting signs of the switch-over points. Another notable feature of this run is that the offline value can be seen to rise and fall. Unlike the Bokhari algorithm, the mGA (and GA) do not necessarily keep the most fit solution in the gene pool. Thus the average of the best solutions in every generation can here be seen to rise and fall. Run data is not available for the mGA runs, however it can subjectively be said to be considerably longer than the comparable Bokhari run in every case.

4.3.3: ThreeD33

The ThreeD33 test was formulated by us to create a test example that would be three-dimensional in nature. This is a block consisting of two interconnected slabs each of 3x3 size. If you visualize only have two rows of an Rubik's cube, then you will see what we have modeled. In this case, the number 33 in the name corresponds to the number of interconnections that exist in the test, not the number of problem nodes. This problem mesh was mapped to a 6x6 machine.

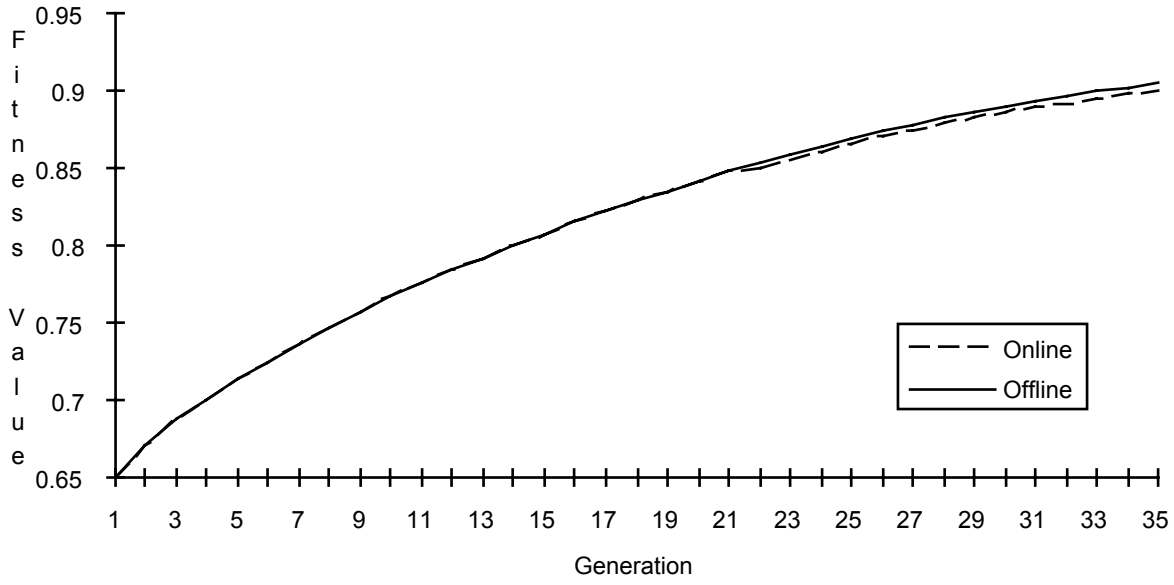

Figure 23: Performance of Bokhari2 Algorithm on ThreeD33

Again, the Bokhari algorithm showed a characteristic performance curve. It was capable of handling this basic three dimensional problem without any great change in performance than the two dimensional problems before.

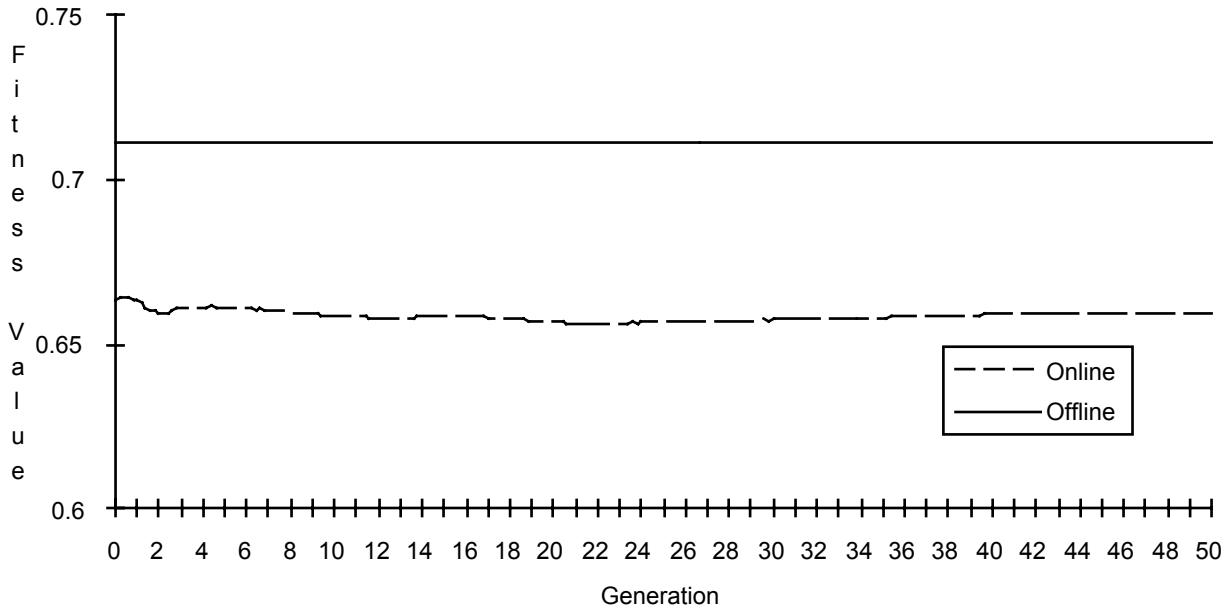

Figure 24: Performance of GA2 Algorithm on ThreeD33

This is perhaps one of the more telling examples of the problems of disruption of schemata in our tests. The genetic algorithm was unable to make any progress against the three dimensional problem set. The initial solution pool provided the best mapping for the entire run, as can be seen by the flat De Jong offline value.

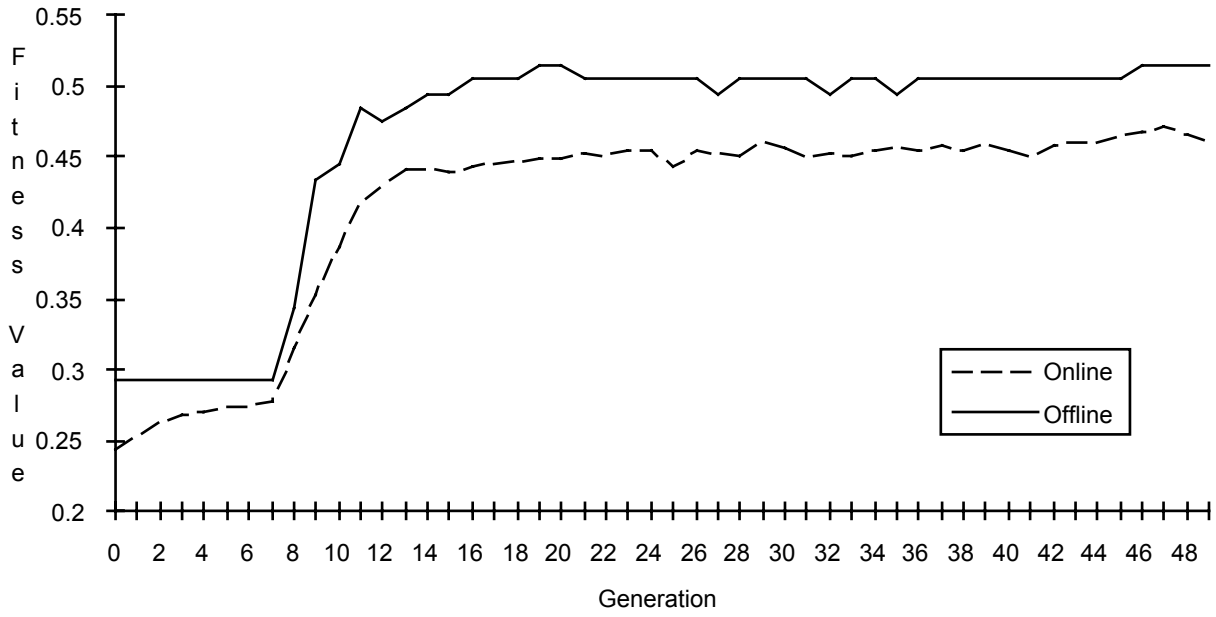

Figure 25: Performance of mGA Algorithm on ThreeD33

The messy genetic algorithm fared even worse against the three dimensional problem. This is unfortunate, since the unordered nature of the messy GA representation offers some potential to keep 3D schemata together. However, perhaps due to the slow build from small building blocks, the mGA was unable to keep these schemata together enough to formulate a good solution.

Chapter 5: Conclusions

Genetic algorithms presented the promise of ameliorating a difficult problem facing the engineering community, namely how to perform FEM analyses on structurally complex problems using SIMD parallel computers. By harnessing the powerful evolutionary force that has produced highly fit solutions to the problems of natural environments, we hoped to produce highly fit solutions to our artificial environment.

An overview of the data we have presented makes the general result fairly clear. Genetic algorithms did not provide the great results that we had envisioned. In practically all cases, the simpler Bokhari process discovered more fit solutions to the mapping problem, and often did so in time superior to the GA and mGA solutions. The messy genetic algorithms proved themselves far too unwieldy to run on serious problems on available equipment.

Why, then, did the ambitions of the genetic algorithms fall short in execution? We believe the answer can be found in the nature of our genetic representation of the mapping problem. Biological genetic material is one dimensional in nature, a strand of DNA. This is well matched to the strand of RNA, and to the serial nature by which DNA and RNA work together to pass biochemical instructions. The mapping problem, however, is not a one-dimensional process. Our mappings have involved two or three dimensional structures that represent volumes, not strands.

The matching of representation to problem is one that we underestimated the gravity of. A genetic algorithm relies upon locality of data, and our genetic operators are specifically created to preserve the adjacency of these effective schemata, assuming a linear representation and problem. Every time that we undertook a genetic operator, we destroyed effective three dimensional schemata far too often, while recognizing too few new good schemata. The messy genetic algorithm has a somewhat better chance of maintaining these schemata in theory, since it is not bound to the linear order of the standard genetic algorithm. It is hard to see any such advantage from our runs, however.

Consider the computer monitor that you are probably familiar with. The image you see upon the screen is composed of hundreds of thin scan lines, firing individual colored pixels upon the screen surface. As a two-dimensional representation, the image is easily recognizable to the viewer. Imagine if one were to capture these scan lines, and stretch them out like a long strand of yarn, dotted with red, green, and blue dots. Laid out in a line, it would not be at all recognizable. The real correlation of the semantics of a video screen is two-dimensional like a grid, not one-dimensional like a string.

In short, we used a tool for a task for which it was not designed. This is not to say that genetic algorithms and messy genetic algorithms are not effective at solving problems. The mass of academic work in this class of methods shows considerable utility in these approaches. Indeed, in the next chapter, we'll discuss how a genetic algorithm could be designed to handle multi-dimensional problems more effectively.

Chapter 6: Future Directions

In our work we have come across several options that future research might take. We have not taken these directions ourselves either due to time restraints, or because they began to leave the scope of the problem we intended to pursue. However, these directions offer promise to make genetic algorithms successful where ours were unsuccessful.

6.1: Parallel Solution of the Mapping problem

This project is attempting to best utilize a SIMD machine for an FEM problem. To do so, we are using a serial machine. Worse, we are running our actual tests on machines that are generally outclassed in the workstation market. It would make a lot of sense to modify this algorithm to run a SIMD machine. The algorithm is highly parallelizable, and the theory is modifiable to make the algorithm fit a SIMD machine even better. For instance, if organisms only bred with neighboring organisms, then chromosome creation could be highly parallelized. We could put as many organisms as would fit on a PE, and then only breed with the organisms of the local PE and its eight neighbor PEs. Fitness testing (suspected to be the major bottleneck in our algorithm) could be highly parallelized, as no inter-PE communication would be needed. Highly fit solutions would appear in spreading pockets that are topographically spread. This would aid in keeping diversity since highly fit (but suboptimal) solutions could not immediately reach the entire organism pool. The advantages to a parallel algorithm for genetic algorithms are promising.

However, in this project we have been concerned with seeing whether genetic algorithms are a reasonable approach to the mapping problem. If this is considered to be viable, then we could quickly proceed to optimize the actual workings of our approach by such methods as parallelization.

6.2: Decompositional Methods

During our work with FEM meshes and the problems of mapping them, we considered a decompositional approach. This approach is based on the heuristic that we should preserve the locality of mesh elements. The problem mesh could be divided into supergroups of elements, each of which would be assigned to a processor or group of processors as a block. These supergroups could then be further broken into subgroups and assigned, and so forth until the final mapping had been made. This approach is an interesting one, but it does not fit in our original proposal to apply genetic algorithms to the mapping problem.

6.3: N-dimensional crossover

Originally brought up in our research by Marton-Erno Balazcs, n-dimensional crossover takes advantage of the argument that all problem solutions are not naturally expressed in a one-dimensional string. In expressing a two-dimensional matrix as a gene-string, for instance, one might create the string by concatenating each row of the matrix. However, the vertical connections of the matrix are then disrupted. Cross-over operators that act on the string cannot then preserve these vertical relations. Some work on the effect on n-dimensional representations on the base theory of genetic algorithms has been done at this point, and will be further explored in papers by Balazcs. N-dimensional representations may have promise in our mapping problem, as no part of our problem is naturally one-dimensional, but perhaps better represented as a 2-D, 3-D, or even k-D, where k is the connectivity of the mesh.

Bibliography

Bokhari, Shahid H., "On the Mapping Problem," *IEEE*, March 1981, pp. 207-214.

Buckles, Bill P., and Fredrick E. Petri, eds., Genetic Algorithms, IEEE Computer Society Press: Los Alamitos, CA 1992.

Davis, Lawrence (eds.), Handbook of Genetic Algorithms, Van Nostrand Reinhold: New York NY, 1991.

Dawkins, Richard, The Blind Watchmaker, WW Norton and Company: New York NY, 1986.

Deb, Kalyanmoy and David E. Goldberg, "mGA in C: A Messy Genetic Algorithm in C", University of Illinois, IlliGAL Report No. 91008, September 1991.

De Jong, Kenneth A., "An Analysis of the Behavior of a Class of Genetic Adaptive Systems", Doctoral Dissertation, University of Michigan, 1975. *Dissertation Abstracts International*, 36(10), 5140B. (University Microfilms No. 76-9381).

Farhat, Charbel, "On the Mapping of Massively Parallel Processors onto Finite Element Graphs," *Computers & Structures*, Vol. 32, No. 2, pp. 347-353.

Garey, M.R. and Johnson, D.S., "Computers and intractability: A guide to the theory of NP-completeness". Freeman, San Francisco, 1979.

Goldberg, David A., "Sizing populations for Serial and Parallel Genetic Algorithms", *The Proceedings of the Third International Conference on Genetic Algorithms*, Eds. J.D. Schaffer., Morgan-Kaufmann, Palo Alto:California, 1989

Goldberg, David A.,"Don't Worry, be Messy", *The Proceedings of the Third International Conference on Genetic Algorithms*, Eds. J.D. Schaffer., Morgan-Kaufmann Publishing:Palo Alto, California, 1989, p 24-30.

Goldberg, David E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley: New York NY, 1989.

Grefenstette, John J. "Optimization of Control Parameters for Genetic Algorithms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, Number 1, 1986, pages 122-128.

Heitkötter, Jörg, "The Hitch-Hiker's Guide to Evolutionary Computation," available via ftp: lumpi.informatik.uni-dortmund.de (129.217.36.140) and on comp.ai.genetic of USENET News.

Hill, Jonathan. "Electromagnetic Analysis using Finite Difference Time Domain Technique and Orthogonal Mesh Generation for Massively Parallel SIMD Architectures". Worcester Polytechnic Institute, 1996.

Holland, John H., Adaptation in Natural and Artificial Systems, University of Michigan Press: Ann Arbor MI, 1975.

Levy, Stephen, Artificial Life, Pantheon Books: New York NY, 1992

Michalson, William R. and R. James Duckworth, "Using Estimated Performance to Optimize Task Allocation in a Distributed Computer System". WPI Technical Report.

Stone, Harold S. High Performance Computer Architecture, Addison-Wesley: Reading, Massachusetts, 1990, pp. 202-441.

Storøy, S and T. Sørevik, "An SIMD, Fine-grained, Parallel Algorithm for the Dense Linear Assignment Problem". Department of Informatics, University of Bergen, Thormøhlensgt. 55, 5020 Bergen, Norway. January 10, 1992

Talbi, E-G. and Muntean, T., "A New Approach to the Mapping Problem: A Parallel Genetic Algorithm". Laboratory de Génie Informatique / Intitut IMAG. BP 53X F-38041 Grenoble, Cedex, France. 1993.

Talbi, E-G. and Muntean, T., "General Heuristics for the Mapping Problem". Laboratory de Génie Informatique / Intitut IMAG. BP 53X F-38041 Grenoble, Cedex, France. 1993.

The following Appendices are supplied on the diskettes provided with this document:

Appendix A: Test Problems

tower25(from Bokhari [1981]), two-dimensional, on 5x5 machine)

mesh33 (from Bokhari [1981], two-dimensional, on 5x5 machine)

threed33 (two-dimensional block, 3x11, on 6x6 machine

elem98 (three-dimensional block, 7x7x2, on 5x5 machine)

elem500 (three-dimensional block, 10x5x10, on 7x7 machine)

Appendix B: Bokhari 2 Code

Appendix C: GA2 Code

Appendix D: mGA Code

Index

Endnotes

[1]CAD/CAE: Computer Aided Design/Computer Aided Engineering
[2]SIMD: Single Instruction, Multiple Data. A further description of the SIMD paradigm will be given.
[3]Bokhari [1981]; Farhat [1989]
[4]MasPar is a trademark of MasPar, Co.
[5]If the number of FEM nodes exceeds the number of processing elements, as we expect it to for most real-world problem, then this will be required. It is also allowable that the number of FEM nodes will exceed the capacity of all of the PEs to hold in memory. In this case the additional FEM nodes would have to be "paged" in and out of active memory, much as modern computers handle virtual memory. Such details are beyond the central thrust of this paper, but are explored in a related work by Jonathan Hill, referenced in the bibliography.
[6]Talbi and Muntean [1993]
[7]Garey and Johnson [1979]
[8]Genetic Algorithms are a subclass of paradigms generally known as "Evolutionary Algorithms", or "Evolutionary Computation". Other subclasses include Evolutionary Programming, Evolution Strategies, Genetic Programming, as well as other new or hybrid paradigms. For a good overview, see Jörg Heitkötter's "The Hitch-Hiker's Guide to Evolutionary Computation," available via ftp: lumpi.informatik.uni-dortmund.de (129.217.36.140) and on comp.ai.genetic of USENET News.
[9]The SimpleGA program is given in the Appendix.
[10]Hill, Jonathan [1996]
[11]Goldberg [1989], pp. 2-10.
[12]Dawkins, Richard [1986], pg. 118.
[13]Holland [1975], pg. 9.
[14]Holland [1975], pg. 11.
[15]Holland [1975]
[16] Holland [1975],
[17]Holland [1975], pg. 74.
[18]Holland [1975]
[19]Note that such ignored information appears in great quantity in biological organisms. See Dawkins [1986], pg. 174 for a discussion of such introns and extrons in biological genetic code.
[20]Goldberg, David A.,"Don't Worry, be Messy", Proc. of the Third International Conference on Genetic Algorithms (1989), Eds. J.D. Schaffer., Morgan-Kaufmann, Palo Alto:California, 1989, p 24-30.
[21]Kalyanmoy Deb and David E. Goldberg, "mGA in C: A Messy Genetic Algorithm in C", University of Illinois, IlliGAL Report No. 91008, September 1991.
[22]Flynn, M.J., "Very High Speed Computing Systems." *Proc. IEEE*, 54, pp. 1901-1909, 1966
[23]Actual implementation of such a solver is the subject of a related paper by Jonathan Hill [1996], also at WPI.
[24]In the case of our MasPar, the controller is a DECstation 3100.
[25]Davis, Handbook of Genetic Algorithms, pg. 13.
[26]In contrast to steady-state reproduction, where one or two individuals are replaced at a time, This is described in the Handbook of Genetic Algorithms, p. 35.
[27]see also comments in the Conclusions section on genetic operators.
[28]Goldberg, David A., "Sizing populations for Serial and Parallel Genetic Algorithms", Proc. of the Third International Conference on Genetic Algorithms (1989), Eds. J.D. Schaffer., Morgan-Kaufmann, Palo Alto:California, 1989
[29]Goldberg, David A., "Don't worry, be messy", [ICGA89], p 24.

[30]Diagram and flag naming conventions are taken from Deb and Goldberg's "mGA in C: A Messy Genetic Algorithm in C" [1991], p 6.
[31]Kalyanmoy Deb and David E. Goldberg, [1991].
[32]Goldberg, Deb, Kargupta, and Harik, "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms" [1993]
[33]Goldberg, Deb, Kargupta, and Harik, [1993]
[34]Goldberg, Deb, Kargupta, and Harik, [1993]
[35]Bokhari, Shahid H., "On the Mapping Problem," *IEEE*, March 1981, p. 214.
[36]Bokhari [1981]
[37]De Jong, [1975]
[38]Goldberg, David A., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, New York:New York, 1989, pp. 106-120
[39]GNU C++ is available for a number of platforms. It is available at ftp.gnu.ai.mit.edu, and copyleft by the Free Software Foundation, Inc.