SOLVING COMPUTATIONALLY EXPENSIVE OPTIMIZATION PROBLEMS USING HYBRID METHODS IN PARALLEL COMPUTING ENVIRONMENTS

John Eddy
Graduate Research Assistant
Design of Open Engineering Systems (DOES) Lab
Dept. of Mechanical and Aerospace Engineering
State University of New York at Buffalo
johneddy@eng.buffalo.edu

Kurt A. Hacker
NASA Graduate Research Fellow
Design of Open Engineering Systems (DOES) Lab
Dept. of Mechanical and Aerospace Engineering
State University of New York at Buffalo
khacker@eng.buffalo.edu

Kemper E. Lewis
Assistant Professor, Corresponding Author
Design of Open Engineering Systems (DOES) Lab
Dept. of Mechanical and Aerospace Engineering
State University of New York at Buffalo
kelewis@eng.buffalo.edu

Keywords: Multidisciplinary Design Optimization, Genetic Algorithms, Hybrid Methods, Parallel Computing

Abstract
The optimization of many realistic large-scale engineering systems can be computationally expensive. The evaluation of a single design configuration can take minutes or hours, and although computing power is steadily increasing, the complexity of the analysis codes continues to keep pace. In this paper we propose a method to utilize parallel processing and hybrid optimization methods to allow for rapid solution to these complex problems. In the first stage of the hierarchical approach developed in this paper, potentially good areas of the design space are identified with a parallel Genetic Algorithm (GA). In the second stage, the best designs within these regions are identified by either heuristic or gradient based optimization techniques. To demonstrate the usefulness of this approach preliminary results are presented from a case study involving the solution of a benchmark optimization problem.

Introduction
In the design of today's increasingly complex engineering systems, the designer is increasingly dependent on computationally expensive computer analysis and simulation codes. Examples of such codes include finite element analysis (FEA), computational fluid dynamics (CFD), heat transfer and vehicle dynamics simulations. The execution time for these types of analyses can be on the order of hours or days for a single function evaluation. This often prevents the application of formal optimization techniques that can require many such evaluations. Thus, the designer is restricted to examining a small subset of the feasible design space likely resulting in a suboptimal design. Despite continuing increases in computing power, the complexity of these analysis codes seems to keep pace with computing advances, making multi-objective, multidisciplinary optimization and concept exploration time consuming to say the least. There has been a great deal of interest of late in the application of distributed computing to problems that were once impractical on serial machines. Presently, it is not uncommon for hundreds of processors to be applied to the solution of a single problem. In addition, due to the decreasing technology costs, supercomputers are no longer limited to a small number of institutions as was the case in years past and can now be accessed by a large portion of the research establishment. Corresponding to this increase in the availability of distributed computers is
the development of numerical optimization algorithms written specifically for the parallel environment. An issue that is not often addressed in the literature is that of computational efficiency, and how the selection of a particular algorithm might be dependent on the length of time a designer can afford to spend on the analysis. For example, if the computational time allotted for a problem is severely constrained, an algorithm that produces steadily improving designs might be chosen in case the optimization has to be prematurely terminated.

In this paper we present the results of ongoing work to develop an approach that provides the designer with multiple good solutions to choose from at a minimum of computational cost. This is important because computational cost is often directly related to the solution time. Since many designers have a limited time in which to solve a problem, long solution times may preclude the use of optimization techniques or even evaluating more than a cursory number of potential solutions. This can have a detrimental effect on the performance of the final design.

In addition to being computationally expensive to solve, complex design optimization problems may have highly nonlinear design spaces with many local optima. Gradient-based techniques, which are very efficient in finding local optima within convex areas of the design space, are not well suited to this type of problem. Conversely, heuristic optimization techniques such as Genetic Algorithms and Simulated Annealing are better suited to finding global solutions in multimodal design spaces but typically require many function evaluations before convergence is achieved.

The approach that we are developing utilizes both gradient based and heuristic optimization algorithms to find good design points more efficiently than either could alone. This combination, known as hybrid or memetic algorithms appears numerous times in the literature. In fact hybrid can be used to describe the merging of some of the properties of any optimization approaches, (i.e. a genetic algorithm and a simulated annealing algorithm). In our work, we are using the combination of GAs and gradient-based methods in a hierarchical procedure run on a distributed memory parallel computer.

Genetic Algorithms are well suited to being executed in parallel. Each processor can contain an independent population of designs that evolves independently or while interacting with the rest of the processors. This is because each member of the population is evaluated individually. Distributed parallel architectures offer the added advantage of flexibility. This gives the designer the ability to run different algorithms on each processor or the same algorithm on all processors.

A hierarchical approach is developed to take advantage of the fact that gradient-based algorithms are very efficient in locating local optima. In the first stage a genetic algorithm is run across multiple processors with the goal of identifying areas of the design space that warrant further investigation. Once these have been identified a test is performed to determine whether a gradient-based optimizer should be used to locate the local optima or if the Genetic algorithm should be reapplied starting from this area. Each of these local searches is then run in parallel. The goal of this approach is two fold - first to efficiently find good designs with a minimum of function evaluations and also to provide the designer with a variety of solutions to choose from.

The remainder of the paper is divided up as follows. In the next section the technical foundation of this work is presented. This includes a discussion on the use of Genetic Algorithms in optimization, parallel optimization techniques, and hybrid methods. Following this, the specific approach being presented in this work is described. Finally the case study we are considering is discussed along with preliminary results and conclusions.

Technical Background

Genetic Algorithms in Optimization

Genetic Algorithms were originally developed to imitate the processes by which living beings evolve. Nearly all design methodologies incorporate the concept of evolving designs. Even designers who do not subscribe to a methodology typically evolve their designs by trial and error.

Genetic Algorithms are most useful for problems involving multimodal design spaces. The fact that they do not consider any gradient information makes it possible for the algorithm to move between the peaks of a multimodal space. A gradient-based optimizer would typically remain within a single mode throughout the solution process likely resulting in a highly suboptimal solution. A suboptimal solution is also possible when using a GA in a multimodal space but it is less likely. It is also likely to be closer to the true optimal than the solution given by the gradient-based optimizer.
The primary focus of our work is on computational efficiency both in memory usage and computational time. We have tailored our Genetic Algorithm accordingly. In the following paragraphs, we describe some of the methods we have employed to help ensure computational efficiency.

**Binary encoding**
Each design variable is represented as a single 32 bit signed integer value within the computer. Therefore, 30 bits are available for genetic encoding. The desired decimal precision for each design variable is input by the user. Floating point numbers are converted to integers by multiplying by $10^{\text{prec(i)}}$ where prec(i) is the desired decimal precision of the i\textsuperscript{th} design variable. An array of 30 short integers for genetic encoding would require at least 60 bytes or 240 bits compared to 4 bytes or 32 bits and accommodations would still be necessary for a sign and a decimal place.

Another advantage of using the signed integer representation of design variables is that conversion from a binary representation to a decimal representation requires no code. It is done at the hardware level which saves a great deal of computational time.

**Dynamic Memory Allocation**
All the arrays in the genetic algorithm are sized at run time using dynamic memory allocation commands according to the problem parameters (number of design variables, number of constraints, etc). The alternative is to allocate arbitrarily large arrays and perhaps use only a portion of them.

We have incorporated the basic components of reproduction and selection. They are presented and described in the following paragraphs.

**Generation of an Initial Population**
The initial population is simply a collection of designs with design variable values taken as random numbers between the design variable bounds. Each design is tested to be sure that it is unique to encourage diversity amongst the initial population.

**Evaluation of Design Fitness**
An objective function value must be computed for each design. The form of the function is not important to the algorithm. Either a call to another software package or an actual analytical function is suitable.

Constraints are accounted for using a penalty term. The penalty term is added to the objective function as shown below to yield the overall design fitness.

$$
\Phi(\bar{x}) = F(\bar{x}) + r_p \, P(\bar{x})
$$

(1)

Where the multiplier $r_p$ is a large value initially and logarithmically decays to a small value over the maximum number of generations$^5$.

The penalty term is in the form of:

$$
P(\bar{x}) = \sum_{j=1}^{m} \max_{j}(g_j(\bar{x}),0) + \sum_{k=1}^{n} \left| h_k(\bar{x}) \right|
$$

(2)

where $m$ is the total number of inequality constraints all taken as less than or equal to zero and $n$ is the total number of equality constraints all taken as equal to zero.

**Reproduction**
Reproduction is accomplished in three stages. First, a mating pool is created based on the pseudo-objective function values. All individuals are represented at least once in the mating pool. Individuals with higher fitness are represented more times making it more likely that they will be given an opportunity to reproduce.

Individuals are randomly chosen from the mating pool. The number of individuals selected is determined by the size of the population and the user input crossover rate.

**Crossover**
Two individuals selected for reproduction are chosen as a mating pair. Each mating pair can produce any even number of offspring (value input by user). The offspring will be referred to as children.

Two types of crossover are available in this GA. The first will be referred to as Simple Crossover and is performed for each design variable as shown in Figure 1 on the following page.

<table>
<thead>
<tr>
<th>Individual</th>
<th>Binary (single DV)</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent 1</td>
<td>11111111111111111111</td>
<td>65535</td>
</tr>
<tr>
<td>Parent 2</td>
<td>00000000010000000000</td>
<td>0</td>
</tr>
<tr>
<td>Child 1</td>
<td>11111111110000000000</td>
<td>65280</td>
</tr>
<tr>
<td>Child 2</td>
<td>00000000011111111111</td>
<td>255</td>
</tr>
</tbody>
</table>

Figure 1: Simple Crossover
The vertical line serves as a randomly selected crossover point. The actual bit switch is accomplished using bitwise logical operators.

The second will be referred to as Arithmetic Crossover and is carried out according to equations 3 and 4 below.

\[
\text{Child 1} = (r)(\text{Parent 1}) + (1 - r)(\text{Parent 2}) \quad (3)
\]

\[
\text{Child 2} = (1 - r)(\text{Parent 1}) + r(\text{Parent 2}) \quad (4)
\]

Where \( r \) is a random number between 0 and 1\(^4\).

**Mutation**

Children from the crossover stage are randomly selected for mutation. There are two types of mutation available in this GA.

The first will be referred to as Random Bit Mutation. In random bit mutation, a design variable is randomly selected as is a bit location for mutation. The selected bit is negated. If it was a 0 it becomes a 1 and visa-versa. The number of mutations is determined by the number of children created, the total number of bits making up all the children, and the user input mutation rate.

The second will be referred to as Random Design Variable Mutation. In this form of mutation, a design variable is chosen at random and reassigned a random value within the upper and lower bounds for that variable. The number of mutations is determined by the number of children, the number of design variables, and the user input mutation rate.

**Insertion of Children into Population**

Once the children have been created and mutated, their fitness is evaluated as described on the previous page. The worst individual in the current population excluding the children is selected. The children are compared to this design one at a time. If the child is a better design, it is inserted in the place of the population member. The population member is inserted into a database of previous designs. This database will be referred to as the graveyard. If the child is worse, it is inserted into the graveyard. The new worst design is then selected from the current population including any inserted children and the process continues until each child has been considered.

Through these simple evolutionary mechanisms Genetic Algorithms can solve very complex problems. In the following section a discussion on the use of distributed computing to make them even more effective is presented.

**The Role of Parallel Computing to increase Flexibility in Optimization**

Parallelism in computers is not a new concept. The earliest computers had separate multiply and divide units that worked simultaneously. In the 1980's the first massively parallel machines were developed which could bring to bear thousands of processors on such complicated problems as weather forecasting, heat transfer, and computational fluid dynamics. The field of engineering design has also seen research and the development of applications that take advantage of the power of parallel computers\(^5\). During the design of a complex engineering system, the designer must make a number of important decisions. These decisions concern both the configuration of the artifact itself as well as the model or models used to describe the performance of the design artifact. In an ideal case, the model used to describe the performance of the artifact is perfectly accurate, allowing the designer to choose the best combination of design parameters to achieve a required performance level. Often, however, the models used to describe the behavior of a design artifact contain simplifications and assumptions. Generally, the more closely a model matches the true behavior of the system, the more computationally expensive it is to run. Thus the designer with limited computational resources may be forced to choose between using an accurate model and thoroughly exploring the design space. Distributed computing can be used to overcome this problem by providing large increases in computing power in a very flexible form.

One area where there has been extensive research is the implementation of Genetic Algorithms in a distributed environment. Genetic Algorithms are pseudo-random search strategies and typically require numerous function evaluations to arrive at a good solution. This typically limits their usefulness for problems involving computationally intensive analysis. There are a number of strategies for applying GAs in parallel. The simplest is termed 'micro-grained' parallelism\(^6\) and is the execution of a serial GA in parallel by dividing up the function evaluations among the processors. This can achieve at most linear speed-up as the number of processor increases. Other techniques utilize the parallel environment to more efficiently explore the design space by maintaining separate sub-populations on different nodes\(^7\). Some researchers term this type of application a 'distributed GA', although 'parallel' and 'distributed' are often used interchangeably. Since
each population is essentially independent, much more of the design space can be explored in a shorter amount of time. This idea is captured in the work of Lin and Goodman’s who introduced the injection island GA (iiGA), in which the best members of one population are injected into other populations to stimulate further performance increases. In this work it is even possible for the ‘islands’ to be using different model resolutions (e.g. good solutions from the low-resolution islands are injected into the high-resolution islands). The injection island approach is implemented in parallel using the GALOPPS toolkit. These strategies, however, do not guarantee the designer the best computational efficiency. This might be possible only through a combination of different optimization algorithms using multiple model resolutions. A useful decision support tool must be flexible enough to accommodate a number of methods and models running concurrently and present the results in a clear and succinct manner.

**Hybrid Optimization**

Researchers have found that although Genetic Algorithms are good at finding good regions in the design space, they perform poorly when it comes to converging to a final solution within this region. A number of hybrid strategies have been implemented to overcome this problem. A hybrid optimization method is one that combines two or more global or local search methods. Typically global optimization algorithms like Genetic Algorithms or Simulated Annealing are combined with local gradient-based methods. The global optimizer is used to find a good region in the design space and the gradient-based method is used to quickly find the ‘best’ solution within that region. Although hybrid methods have been successfully applied there remain a number of open research issues such as how to most effectively implement them in parallel, and how to handle multiple subsystems and designers.

In this paper we present an approach that takes the idea of the parallel GA and combines it with a hybrid approach. In the first stage of the optimization process a parallel GA is used to perform a low fidelity search of the design space. The purpose of this search is two-fold, to identify good regions of the design space and to characterize these regions as being unimodal or multimodal.

The goal of the preliminary search is to cover as much of the design space as possible, and to that end there are a number of ways to design the parallel GA. Figure 2 on the next page illustrates these four designs labeled a, b, c, and d. Members of the population are indicated by circles or triangles and can be matched with the four processors on which the populations reside.

1. One GA is run over all the processors with a single population. The processors simply share the load of evaluating the potential designs (a).
2. A sub-population is maintained on each node. The initial population is randomly seeded throughout the design space. The bounds on sub-population are equal to the bounds on the design variables (b).
3. A sub-population is maintained on each node. The design space is partitioned and distributed to separate nodes. The initial population is generated from within each local area of the design space, but the individual GAs are allowed to evolve to any area in the total design space (c).
4. A sub-population is maintained on each node. The design space is partitioned and distributed to separate nodes. The initial population is generated from within each local area of the design space, and the individual GAs are not allowed to move out of this local area (d).

The hope in using designs two, three, and four is that a better solution will be reached faster and with less function evaluations than with design one and that the design space will be thoroughly covered so that no pockets of good designs are missed. The flexibility to use any of the above approaches is what makes the parallel computer such a powerful optimization tool.

Once good areas of the design space have been identified these may serve as the starting points for gradient search methods which can efficiently locate local optimal points. There is no guarantee however that the initial search successfully converged to only one local optimal point. Each of the final subpopulations could straddle multiple convex regions. In such cases the method of choice is to rerun a Genetic Algorithm in the local area identified in the preliminary search. This is required in order to have a better chance of locating the true optimal design.

The flexibility of the parallel computer allows both gradient-based and Genetic Algorithms to run simultaneously allowing a designer to rapidly explore the design space.
Figure 2: Four Possible Designs for a Parallel GA

To determine whether a heuristic or gradient-based method is appropriate, some characteristics about the local area of the design space need to be determined. One good source of information is contained in the final population of designs on each of the processors. By calculating the relative homogeneity of the populations we can get a good idea of whether there are multiple local optima located within this local region of the design space. Figure 3 illustrates this idea. In the top portion of the figure all of the members of the population are located in a single minimum, the values of both the design variables and the objective function are similar. In the center portion of the figure is an area of the design space with two local minima, one better than the other. In this case both the design variables and objective function values are different. The bottom part of the figure also shows two local minima, but this time they are of equal magnitude. In this case the designs have similar objective function values but different $x$ values.

To quantify the relative homogeneity of the population the variance of the final population's objective function and design variables can be calculated. A high variance might be an indication that there are multiple local optima present in the area and that a restart of the GA from this area is warranted. Very low variance could be an indication that there is a single local minimum in the area, warranting the use of a gradient-based method.

Figure 3: Distribution of the Final Population of a Genetic Algorithm
Although the variance of the population is certainly not a definitive measure of how many local minima are in a particular area, it is inexpensive to calculate providing the designer with valuable decision support at low cost.

In the following section, we present in detail the specific approach.

**A Hybrid Approach to Parallel Optimization**

In this section details regarding the specific implementation of the hybrid optimization procedure introduced in this paper are presented. A 64 processor Sun Cluster from the University at Buffalo's Center for Computational Research is used for the analyses.

The specific steps in the application of our approach are as follows.

1. Initialize the Genetic Algorithm with problem information including the objective function, constraint information, and bounds on the design variables on each processor.
2. If the designer chooses to do so, partition the design space into equal size sections depending on the number of processors being used. The partitioning occurs by dividing the ranges of one or two design variables among the processors. If no partitioning is used, each processor has the same bounds as the overall problem. The partitions may also be used simply to generate distinct initial populations on each of the processors.
3. The population is randomly generated in one of two ways.
   a. The population on each of the processors is generated randomly from within the local bounds from step 2.
   b. The initial population is generated randomly from within the entire design space.
4. The GAs are executed in parallel in the preliminary search. Once all nodes are completed the variance of the final population on each node is computed. The results are sent to a central processor where they are compiled and presented to the designer.
5. On the designer’s discretion, a Sequential Linear Programming (SLP) algorithm or the GA is started from the best design point in the initial search.

Automating this process is the subject of ongoing research.

In the next section we describe the results of a case study used to evaluate the performance improvements possible through this approach.

**Case Studies**

In this section we present a case study used to verify the effectiveness of our approach. Our focus is on finding good solutions to complex design problems with multiple local optima with a minimal number of function evaluations. This problem, defined below in equations 5 through 8, was developed by Professor Andy Keane at Southampton University:

Minimize:

$$F(x) = \sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)$$

Subject to:

$$g_1(x) = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$

$$g_2(x) = \sum_{i=1}^{n} x_i - \frac{15n}{2} \leq 0$$

$$0 < x_i < 10 \quad i = 1, n$$

Where \( n \) is the number of design variables. This problem becomes increasingly multimodal as the number of design variables increases.

This problem was designed as a benchmarking tool for measuring the performance of Genetic Algorithms. It is similar to real world design problems in that for large \( n \) it is multivariate, multimodal, and highly nonlinear.

To see how this hybrid approach works for problems of increasing modality, the problem is solved with \( n \) values of 10, 20, and 50. The results are compared with previous results to verify the accuracy of the solution, and the efficiency of the solution process is measured against results from executing only the Genetic Algorithm.
This part of the results consists of verifying that the GA returns values of similar magnitude as previous studies using this problem. Figure 4 shows the evolution of four independent subpopulations for n=20. The GA was terminated at 200 generations, which is equivalent to approximately 12500 function evaluations. The minimum value of the objective function at this point occurs on processor 4 and is equivalent to -0.70. This compares favorably with that of previous studies that reached -0.76 after 20000 function evaluations.

In Figure 5 is shown the effect of using the local search from the point the preliminary search is stopped. The Sequential Linear Programming (SLP) algorithm is executed for the point where the GA is stopped. It can be seen that a large improvement in the objective function occurs in only a few function evaluations. Also plotted are the results obtained if the GA is allowed to continue to run. As expected the GA continues on a rather flat trajectory improving far less that the gradient based SLP. The reason for the rapid performance gains possible using a gradient-based local search algorithm can be clearly understood from Table 1. After the GA completes the preliminary search (Stage 1), neither of the constraints is active. In Stage 2, the local search quickly drives $g_2$ to zero. This result confirms the idea that although GAs are effective at exploring new areas of the design space they perform poorly in isolating the best design from within a local area.

![Figure 4: Evolution of Subpopulations for n=20](image)

From Figure 4, the rate of decrease of the objective function is decreasing rapidly when the GA was stopped. In other words, the number of function evaluations needed for a unit decease in $F(x)$ is increasing.

![Figure 5: Effect of Local Search on Objective Function Value for n=20](image)

### Table 1: Results from n=20

<table>
<thead>
<tr>
<th>Node</th>
<th>Function Evaluations</th>
<th>Variance in Final Pop</th>
<th>$F(X)$</th>
<th>$g_1$</th>
<th>$g_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12554</td>
<td>1.60E-05</td>
<td>-0.65</td>
<td>-0.30</td>
<td>-109.61</td>
</tr>
<tr>
<td>2</td>
<td>12551</td>
<td>6.00E-06</td>
<td>-0.60</td>
<td>-0.21</td>
<td>-114.05</td>
</tr>
<tr>
<td>3</td>
<td>12553</td>
<td>3.80E-05</td>
<td>-0.72</td>
<td>-0.02</td>
<td>-116.32</td>
</tr>
<tr>
<td>4</td>
<td>12549</td>
<td>1.60E-05</td>
<td>-0.65</td>
<td>-0.02</td>
<td>-113.29</td>
</tr>
</tbody>
</table>

The results in Tables 2 and 3 show similar trends for n=50 and n=10. They list the values of the objective function, constraints, variance of the objective function in the final population, the number of function evaluations, and the percentage change in the value of the objective function between Stages 1 and 2. The results have a common theme - relatively large decreases in the value of the objective function for a small number of additional function evaluations. The improvements are more dramatic for larger n. One theory for this is that when n is large only a small part of a given local minima can be explored by the population. This results in a solution that is far from the local optimal design. This allows for large improvements when the gradient-based local search
is performed. For small \( n \) there is a better chance that the population explores a point close to the local optimal.

<table>
<thead>
<tr>
<th>( n=50 )</th>
<th>STAGE 1: GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Function Evaluations</td>
</tr>
<tr>
<td>1</td>
<td>12600</td>
</tr>
<tr>
<td>2</td>
<td>12600</td>
</tr>
<tr>
<td>3</td>
<td>12600</td>
</tr>
<tr>
<td>4</td>
<td>12600</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( n=50 )</th>
<th>STAGE 2: SLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Function Evaluations</td>
</tr>
<tr>
<td>1</td>
<td>4047</td>
</tr>
<tr>
<td>2</td>
<td>7377</td>
</tr>
<tr>
<td>3</td>
<td>3167</td>
</tr>
<tr>
<td>4</td>
<td>3439</td>
</tr>
</tbody>
</table>

**Table 2: Results for \( n=50 \)**

<table>
<thead>
<tr>
<th>( n=10 )</th>
<th>STAGE 1: GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Function Evaluations</td>
</tr>
<tr>
<td>1</td>
<td>12318</td>
</tr>
<tr>
<td>2</td>
<td>12101</td>
</tr>
<tr>
<td>3</td>
<td>12106</td>
</tr>
<tr>
<td>4</td>
<td>12371</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( n=10 )</th>
<th>STAGE 2: SLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Function Evaluations</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>123</td>
</tr>
<tr>
<td>3</td>
<td>176</td>
</tr>
<tr>
<td>4</td>
<td>194</td>
</tr>
</tbody>
</table>

**Table 3: Results for \( n=50 \)**

**Table 4: Effect of Partitioning the Design Space**

With partitioning, the objective function is lower, which seems to support the idea that if there is a smaller design space to consider a more thorough search can be conducted. In future work the design space will be partitioned along more that one design variable making possible more solid conclusions.

The final issue to consider is the use of the variance to give the designer insight into which method to use in Stage 2 of the approach. The results to this approach were inconclusive. The variance of the final population was calculated for number of trials and generally speaking was low, around the order of 1.0xE-04. Trials using the Genetic Algorithm in stage 2 of the approach were performed for the cases where the variance was highest. On restart, however, the GA took many more function evaluations than the SLP algorithm to reach the same level of improvement. We are currently investigating the use of the variance of the design variables in combination with the variance of the objective function to give us more insight into what type of algorithm is most appropriate for Stage 2.

**Conclusions and Future Work**

In this paper we present a hybrid optimization approach implemented on a distributed computing environment. Preliminary results have shown that good increases in efficiency can be achieved with its use over the use of a Genetic Algorithm only. Initial results from partitioning the design space to improve the efficiency of the solution process are also presented. In addition we explore the idea of using variance to determine the most efficient local search.
algorithm. Listed below are some of the features and capabilities we intend to include in our algorithm in the future.

- Increased number of algorithms for both the initial search and the second phase which involves a more detailed exploration of specific areas of the design space. Other algorithms include Simulated Annealing, Random Search, Grid Search, and other gradient-based approaches.
- Increased automation of the algorithm. We intend to automate the transition from phase 1 to phase 2 while allowing the designer to intervene whenever he/she desires.
- Sharing of design information amongst processors. We will allow each processor access to the other processors design database to help prevent evaluation of duplicate designs.
- Improved methods of dividing the design space for exploration by different processors. The algorithm currently divides the space according to a single design variable.
- Incorporation of design injection between processors to hasten the evolution of individual populations.

Acknowledgments

We acknowledge the support of NASA-Langley Research Center (NGT152185) and the National Science Foundation (grants DMI-9800435 and DMI-9875706) in this work. We also acknowledge the support of the Center for Computational Research at the University at Buffalo for the computing time.

References


