Solving of Discrete Multiobjective Problems using an Evolutionary Algorithm with a Repair Mechanism

Jesse B. Zydallis and Gary B. Lamont
Dept of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433, USA

Abstract—**The solving of Real-World Multiobjective problems (MOPs) with an Evolutionary Algorithm (EA) is an increasing area of interest. Presented in this paper is the application of a building block based EA to a real-world discrete MOP. A constraint handling method had to be designed and employed. The description of this method and the repair mechanism instrumented to repair infeasible solutions is described in detail along with statistical analysis.**

I. Introduction

Real-World optimization problem solving is a task that we as engineers and scientists encounter daily. Having a good tool and validating the ability of that tool to solve optimization problems that are important to our work is an integral part of being an engineer or scientist. In this paper we present a a population based, stochastic approach, an Evolutionary Algorithm (EA), to solving a complex, discrete multiobjective problem (MOP) with equality constraints.

The algorithm utilized to solve this MOP is the Multiobjective Messy Genetic Algorithm II (MOMGA-II) [7]. This algorithm is unique in its explicit manipulation of Building Blocks (BBs) and variable string lengths to find the solution to complex MOPs. This paper is organized as follows: The following section presents background information on the MOMGA-II. Section III. presents the problem domain information followed by the results and conclusions.

II. MOMGA-II

The MOMGA-II is a modification of the MOMGA [6] created by Van Veldhuizen and Lamont. The modifications of the MOMGA resulted in an increase in efficiency, and the ability to solve discrete MOPs and MOPs containing any number of input decision variables [7]. The MOMGA-II modifications reflect the improvements completed to the messy GA [4] to create the fast messy GA [3]. The fast-messy GA and MOMGA-II consist of the following phases: *Initialization Phase, Building Block Filtering, Juxtapositional Phase* [3], [7]. The difference from the MOMGA lies in the Initialization phase and the Primordial phase, which is referred to as the Building Block Filtering (BBF) phase. The initialization phase creates the population randomly and uses a reduced population size.

The MOMGA-II utilizes a probabilistic approach in initializing the population. This is referred to as Probabilistically Complete Initialization (PCI) [3]. The probabilistic BB approach initializes the population by creating a controlled number of BB clones of a user specified size. Building blocks are sub-strings that contain "good" information, i.e. an example of a BB of size two is one in which a 1 in position 0 and 10 typically results in a good solution. This approach effectively reduces the computational bottlenecks encountered with the original MOMGA's initialization phase by reducing the size of the initial population required.

After creation these BBs are filtered, through a Building Block Filtering (BBF) phase, to probabilistically ensure that all of the desired BBs are in the initial population. This filtering is accomplished through a schedule consisting of the random deletion of bits in each of the chromosomes throughout the user specified input schedule. The schedule specifies the generations to conduct the random deletion, the number of specified bits to delete from the chromosomes and the number of juxtapositional generations to execute. The random deletion of bits is alternated with tournament selection between the building blocks that have been found to yield a population of "good" building blocks. This complex parameter selection is a very critical element of the MOMGA-II. A more detailed fmGA discussion of the theoretical analysis of the schedule is presented in [3], [5].

In order to accomplish this filtering operation, all of the BBs must be evaluated with respect to the fitness function. Since the BBs are in actuality only partial strings, also referred to as underspecified population members, a method must exist to complete this evaluation. A competitive template is utilized here. This template is a fully specified population member, i.e. a member that is equal in length to the specified string length. To complete the evaluation, the underspecified population member's bits replace the respective bits in the template and then the evaluation takes place. This effectively allows the BB to be evaluated based on the bits present in the BB.

During the Juxtapositional phase the building blocks that have been found through the Initialization phase and the BBF phase are recombined through the use of a cut and splice op-

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

 0-7803-7150-X/01/$10.00@2001 IEEE

erator alternated with tournament selection with thresholding. The cut-and-splice operation takes the population members from the BB size and combines them to reach the specified string length. This process increases the length of the individual's strings towards becoming fully specified while keeping the best members through the selection operator. Again the fitness evaluation is completed on fully specified individuals, through the use of the competitive templates if necessary.

These modifications have shown the MOMGA-II to be a statistically more efficient algorithm than the MOMGA, while reutilizing much of the same code as the original MOMGA. In terms of the competitive template, the MOMGA-II operates in the same fashion as the MOMGA. Each of the strings in the population are evaluated with respect to the different functions that one is attempting to optimize. The different competitive templates are used to fully specify individuals that have unspecified bits. In the case of overspecification, where a gene location has multiple allele values, a left to right scheme is used upon which the first value encountered to specify the gene becomes the allele value for that gene.

III. Multiobjective Problems and Terminology

MOPs typically consist of competing objective functions, which may be independent or dependent on each other. An example of this is a company's quest to purchase a backbone for their network that will provide the greatest throughput at the least monetary cost. These objectives are are highly dependent on each other as increased cost will bring about increased throughput and vice-versa. Although single-objective optimization problems may have a unique optimal solution, MOPs usually have a possibly uncountable set of solutions, which when evaluated produce vectors whose components represent trade-offs in decision space.

Pareto optimal solutions are those solutions within the search space whose corresponding objective vector components cannot be all simultaneously improved. These solutions are also termed *non-inferior, admissible*, or *efficient* solutions. Their corresponding vectors are termed *nondominated*; selecting a vector(s) from this vector set implicitly indicates acceptable Pareto optimal solutions (*genotypes*). These solutions may have no clearly apparent relationship besides their membership in the Pareto optimal set. It is simply the set of all solutions whose associated vectors are nondominated; we stress that these solutions are classified as such based on their *phenotypical* expression. Their expression (the nondominated vectors), when plotted in criterion (*phenotype*) space, is known as the current *Pareto front*.

The concept of Pareto Optimality is integral to the theory and analysis of MOPs. A way to determine if one solution is "better" than another is a necessity here as well as in all problems. Pareto concepts allow for the determination of a set of optimal solutions in MOPs. Some key Pareto concepts are defined mathematically [6]. The following Pareto notation is utilized in order to clarify the analysis. P_{known} and P_{true} are genotype sets of Pareto optimal solutions. Each of these partially ordered sets has a corresponding phenotype set, or Pareto front set termed PF_{known} and PF_{true}. P_{known} represents the Pareto optimal solutions that the particular Multiobjective Evolutionary Algorithm (MOEA) has found versus P_{true} which contains the true Pareto optimal solutions for the MOP, PF_{known} and PF_{true} are the representative phenotype sets.

IV. Constraint Handling

Discrete Optimization problems with hard constraints are difficult problems to solve with MOEAs. This is especially true if the constraints drastically reduce the size of the feasible solution space. An easy method to implement would be to allow the MOEA to execute as normal and remove the infeasible solutions at the end. This is a simplistic method to implement but suffers from the possibility of producing no solutions to the problem if the constraints are very restrictive. What may occur is that the MOEA finds solutions to the unconstrained problem and thereby throws away solutions that actually meet the constraints. This would occur in the Pareto dominance routine since at each generation the Pareto optimal solutions that the MOEA found for the MOP may change. A simple example of this is if the MOEA found a set of Pareto optimal solutions in the first generation that meet the constraints but the Pareto optimal solutions found in the 5th generation do not meet the constraints and dominate those found in the 1st generation. This would result in a solution set of infeasible solutions.

Another approach to the constraint handling problem is presented by Deb [1], [2]. He proposed a constraint handling method that is utilized within the Pareto dominance selection mechanism, presented in Definition 1, and compared it to other methods that exist. He states that the method presented in his paper produced "better" results than other methods on the limited test problems utilized.

Definition 1 Pareto Based Constraint Handling Selection
In comparing two solutions, i and j:

1. *If both solutions, i and j, are feasible, choose the solution with the "better" value.*
2. *If solution i is feasible and solution j is not, choose solution i.*
3. *If neither solution, i or j, is feasible, but solution i has a smaller overall constraint violation, choose solution i.*

The aforementioned constraint handling methods were considered for inclusion into the MOMGA-II. In order to accomplish this, each of the phases must be analyzed to determine if the method can be effectively applied in this MOEA. In the initialization phase the population of the MOMGA-II is randomly created. This initialization phase implementation attempts to disperse the population members across the search

space. There is a necessity in creating population members that are sufficiently different from each other in attempting to find various "good" building blocks. The disadvantage of this approach is apparent once a highly constrained MOP (HC-MOP) is encountered. In a HCMOP, the majority of solutions, if not all solutions, created by this random process are infeasible. This is a poor starting point for the MOMGA-II. Since the MOMGA-II relies on the idea that the PCI initialization phase probabilistically produces the majority of "good" BBs, a population of infeasible solutions potentially has **0** "good" BBs, or may lead to all of the feasible BBs being lost in the BBF phase. This necessitates a different initialization method.

A population of infeasible members also illustrates problems associated with Deb's and other's constraint handling methods when applied to a BB based EA [1], [2]. With infeasible solutions and the MOMGA-II's explicit BB based manipulation and selection routines, the MOMGA-II has a low probability of producing a feasible BB or population member in this situation. This is a result of the random initialization and selection mechanism. In any problem a random solution method may or may not produce a "good" solution or BB. The selection mechanism utilized promotes population members that have the best fitness, not necessarily those that are feasible. The MOMGA-II proceeds to solve the MOP with the best BBs it finds, feasible or infeasible. One may then ask, *Why not just keep and use the feasible BBs in the BBF and selection routines?* This question assumes that there are some feasible BBs to start with and that there is enough diversity in the population members to prevent premature convergence. While this situation may be true for MOPs with "relaxed" constraints, this generally is not true for HCMOPs. HCMOPs may start with **0** feasible population members and BBs. If this is the case, other constraint handling methods generate solutions but do not reverse the affect of the BBF phase and its removal of "poor" solutions from the population, which in fact may be feasible solutions.

Another factor that must not be overlooked is the competitive template. Even with the assumption of a feasible template to start with, feasible or infeasible BBs may lead to an infeasible population member. This is important since the competitive template has a large affect on the population members while the string lengths are short, since more bits are used from the template. As the strings increase in length in the juxtapositional phase, the competitive template has a lesser affect on the evaluation and possibly no affect once the member is fully specified. The issue here is that even keeping a feasible template does not guarantee a feasible population member. This is due to the combination of the BBs and the templates for evaluation and the different possible combinations of BBs in producing a final solution.

To avoid the potential of producing few feasible solutions, the simple post-constraint handling method was deemed inappropriate as well as the method suggested by Deb. The approach used in the MOMGA-II does not rely on constraint handling being coupled with the selection mechanism but instead relies on a repair mechanism. The use of a repair mechanism allows the algorithm to proceed without modification to the existing selection routines.

V. Repair Mechanisms

The MOMGA-II is a unique MOEA in comparison to most other MOEAs in its building block manipulation, variable string lengths and competitive templates. In order to effectively handle repairing population members, there are a number of issues that must be addressed. Repair of population members appears to be the most effective means of solving HCMOPs with the MOMGA-II. This section discusses potential repair mechanisms, potential problems with these mechanisms and methods chosen for implementation.

The initialization of the population and the competitive template are a key issue in the effectiveness of the MOMGA-II. Since HCMOPs tend to yield few feasible population members the proposed solution is to randomly generate the population members and the competitive templates. Immediately following this repair all of the population members and competitive templates. This ensures that the MOMGA-II starts with a feasible population and templates.

The repair of the population members is necessary. A repair mechanism must be decided upon as well. The repair mechanism presented here assumes an integer based decision variable space MOP. In an integer based input variable space, the following procedure repairs the population members:

1. Randomly choose a bit position to start at.
2. Randomly choose a direction to start with.
3. Proceed in the specified direction to increment or decrement each decision variable by one unit until the member is feasible

This method has overhead associated with the constant analysis to determine if the population member is feasible or not. The advantage of this method is that one decision variable is not decremented by a huge amount while the others are left the same. The idea here is to "move" the population member by a uniform amount in each decision variable dimension until a feasible solution is achieved.

Once the repair mechanism completes, all of the population members are feasible going into the BBF phase. While in the BBF phase the population members are allowed to become infeasible. If during the BBF phase, each BB was repaired, this would occur with respect to the specific competitive template that is being used. Essentially it may be the template that is causing the combination of the BB and the template to be infeasible. Therefore the BBs are allowed to be infeasible and no repairs take place in this phase.

The juxtapositional phase proceeds with potentially infeasible BBs and a feasible template. As each generation of this phase executes, the population members are not repaired following each cut-and-splice operation. This is to help avoid convergence to a non-optimal solution. However, after each generation of the juxtapositional phase, all of the Pareto front population members are recorded to a data file. The members that are written to the data file are repaired to ensure that the MOMGA-II produces a feasible solution set. As the juxtapositional generations are completed, the population member's strings become longer. The repair mechanism is not used to allow for infeasible solutions to exist. These infeasible solutions may lead to feasible solutions in subsequent recombination generations and therefore are not removed. Once all three phases of the algorithm complete, all of the population members are repaired and the "best" members become the new competitive templates for the next iteration of the phases.

VI. Results

The MOMGA-II's repair mechanism is applied to a discrete HCMOP consisting of 15 decision variables, 3 minimization objective functions and strict equality constraints. The results with and without the repair mechanism are presented. The objective functions are:

$$\begin{aligned} S = &-(0.8x_{1,1} + 0.3x_{1,2} + 0.6x_{1,3} + 0.001x_{1,4} + 0.001x_{1,5} \\ &+0.4x_{2,1} + 0.8x_{2,2} + 0.6x_{2,3} + 0.001x_{2,4} + 0.001x_{2,5} \\ &+0.001x_{3,1} + 0.001x_{3,2} + 0.1x_{3,3} + 0.8x_{3,4} + 0.4x_{3,5}) \end{aligned} \quad (1)$$

$$\begin{aligned} W = &20.2(x_{1,1} + x_{2,1} + x_{3,1}) + 28.5(x_{1,2} + x_{2,2} + x_{3,2}) \\ &+35.7(x_{1,3} + x_{2,3} + x_{3,3}) + 19.9(x_{1,4} + x_{2,4} + x_{3,4}) \\ &+22.5(x_{1,5} + x_{2,5} + x_{3,5}) \end{aligned} \quad (2)$$

$$\begin{aligned} V = &1650(x_{1,1} + x_{2,1} + x_{3,1}) + 2475(x_{1,2} + x_{2,2} + x_{3,2}) \\ &+2887.5(x_{1,3} + x_{2,3} + x_{3,3}) + 1705(x_{1,4} + x_{2,4} + x_{3,4}) \\ &+2200(x_{1,5} + x_{2,5} + x_{3,5}) \end{aligned} \quad (3)$$

subject to:

$$(x_{1,1}, \ldots x_{3,5}) \geq 0 \quad (4)$$

$$(x_{1,1}, \ldots x_{3,5}) \in I \quad (5)$$

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} = 10 \quad (6)$$

$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,5} = 5 \quad (7)$$

$$x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} + x_{3,5} = 1 \quad (8)$$

Figure 1 presents the true Pareto Front, found through a total enumeration of the search space, versus the Pareto Front the MOMGA-II with its repair mechanism finds. We see that the MOMGA-II with the repair mechanism finds all of the points! In the case of the MOMGA-II without the repair mechanism, none of the true Pareto Front points are found since it does not even find one feasible point. This illustrates how useful a repair mechanism is with HCMOPs.

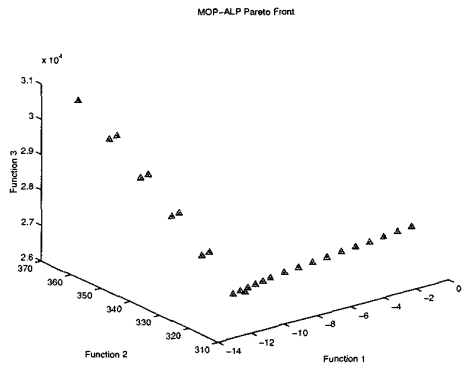

Fig. 1. True Pareto Front Versus MOMGA-II Pareto Front

VII. Conclusions

Through testing we have shown that our constraint handling method for messy GAs is effective on the test problem presented here. Various constraint handling techniques exist and are employed today but they are not the most effective techniques for messy GA based processing. Future work will look at additional problems and the probabilities associated with messy constraint handling techniques. Further the work presented here can be expanded to non-integer based decision variables since these variables are discretized in the computer's representation.

References

[1] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[2] Kalyanmoy Deb and T. Meyarivan. Constrained Test Problems for Multi-Objective Evolutionary Optimization. KanGAL report 200005, Indian Institute of Technology, Kanpur, India, 2000.

[3] David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, CA, July 1993. Morgan Kaufmann Publishers.

[4] David E. Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[5] Hillol Kargupta. *SEARCH: Polynomial Complexity, And The Fast Messy Genetic Algorithm.* PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[6] David A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, May 1999.

[7] Jesse B. Zydallis, David Van Veldhuizen, and Gary Lamont. A statistical comparison of multiobjective evolutionary algorithms including the MOMGA-II. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001)*, volume 1993 of *Lecture Notes in Computer Science*, pages 226–240, Berlin, 2001. Springer-Verlag.