Multiobjective Optimization with Messy Genetic Algorithms

David A. Van Veldhuizen
Air Force Research Laboratory
Optical Radiation Branch
8111 18th Street
Brooks AFB, TX 78235-5215
david.vanveldhuizen@hedo.brooks.af.mil

Gary B. Lamont
Dept. of Electrical & Computer Engineering
Graduate School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-7765
gary.lamont@afit.af.mil

ABSTRACT

This paper extends the traditional notion of *building blocks* as found in single-objective Evolutionary Algorithms (EAs) to the Multiobjective Optimization Problem (MOP) domain. This effort aims to develop more effective and efficient Multiobjective EAs (MOEAs) by using relevant MOP concepts and a consistent MOEA-Pareto notation to depict building block processing in solving MOPs. Specifically, an innovative extension of the building block-based messy genetic algorithm (called the MOMGA) is presented and applied successfully to our MOP test suite. This approach is shown to be quite effective when compared to other contemporary MOEAs.

1. INTRODUCTION

Multiobjective Evolutionary Algorithms (MOEAs) are now a well-established field within Evolutionary Computation, initially developed in 1984 when the first MOEA designed to solve Multiobjective Optimization Problems (MOPs) was implemented. Over 285 published papers have since proposed various MOEA implementations and applications, and to a much lesser extent, underlying MOEA theory [1; 3; 19].

Our research centers around solving scientific and engineering MOPs with MOEAs; an additional goal is developing and defining further MOEA theoretical contributions. Solving MOPs with MOEAs presents several unique challenges (e.g., vector-valued fitness, *sets* of solutions, evolutionary operator complexity). For a good introduction to the relevant issues and past EA-based approaches, see Van Veldhuizen and Lamont [21], Coello Coello [4], and Fonseca and Fleming [7]. This paper focuses on the traditional notion of *building blocks* as found in Genetic Algorithms (GAs), extending the concept to the MOP domain in an effort to develop more effective and efficient MOEAs. This is an innovative MOEA approach [19; 21].

The remainder of this paper is organized as follows. Sections 2 and 3 introduce relevant MOP and building block concepts. Section 4 addresses the consequences of extending an existing building block-based EA to the MOP domain. Sections 5 and 6 describe our MOEA's implementation and initial experimental results, while Section 7 presents our conclusions and recommendations for future work.

2. MOP BACKGROUND

Although single-objective optimization problems may have a unique optimal solution, MOPs (as a rule) present a possibly uncountable set of solutions, which when evaluated produce vectors whose components represent trade-offs in decision space. A decision maker then implicitly chooses an acceptable solution by selecting one of these vectors. In mathematical terms, an MOP optimizes the components of a vector $F(\vec{x})$, where $\vec{x}$ is an n-dimensional decision variable vector ($\vec{x} = x_1, \ldots, x_n$) from some universe Ω. Or in general,

$$\begin{array}{ll} \text{optimize} & F(\vec{x}) = (f_1(\vec{x}), \ldots, f_k(\vec{x})) \\ \text{subject to} & g_i(\vec{x}) \leq 0, i = 1, \ldots, m,\ \vec{x} \in \Omega\ . \end{array} \qquad (1)$$

An MOP thus consists of n variables, m constraints, and k objectives, of which any or all of the objective functions may be linear or nonlinear [15]. The MOP's evaluation function, $F : \Omega \rightarrow \Lambda$, maps the decision variable(s) to vectors.

We have defined an MOP's global optimum to be the *Pareto front* determined by evaluating each member of the *Pareto optimal solution set* [19; 20]. Since Pareto concepts are crucial components of the MOP domain and MOEA implementations, they are now briefly described.

2.1 Pareto Concepts and Notation

Pareto optimal solutions are those solutions within a 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* [13]; selecting a vector(s) from this nondominated vector set implicitly indicates an acceptable Pareto optimal solution(s) (*genotype(s)*). 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 *Pareto front*. Some researchers have erroneously used these terms in the MOEA literature, suggesting that a consistent notation is required. An MOEA's structure can easily lead to confusion when identifying Pareto components, also driving the need for a consistent, understandable terminology. During MOEA execution, a "local" set of Pareto optimal solutions (with respect to the *current* MOEA generational population) is determined at each EA generation and termed $P_{current}(t)$, where t represents the generation number. Many MOEA implementations also use a secondary population, storing all/some Pareto optimal solutions found through the generations [19; 21]. Because a solution's classification as Pareto optimal depends upon the context within which it is evaluated (i.e., the given set of which it's a member), corresponding vectors of this set must be (periodically) tested, removing solutions whose associated vectors are dominated. We term this secondary population P_{known} (t). This term is also annotated with t (representing completion of t generations) to reflect possible changes in its membership during MOEA execution. P_{known} (0) is defined as $\emptyset$ (the empty set) and P_{known} alone as the *final*, overall set of Pareto optimal solutions returned by an MOEA.

Different secondary population storage strategies exist; the simplest is when $P_{current}$ (t) is added at each generation (i.e., $P_{current}$ $(t) \bigcup P_{known}$ $(t-1)$). In this case P_{known} (t) is thus the set of Pareto optimal solutions *yet found by the MOEA through generation t*. Of course, the *true* Pareto optimal solution set (termed P_{true}) is not explicitly known for MOPs of any difficulty. P_{true} is defined by the functions composing an MOP; it is fixed and does not change (see Figure 1). Because of the manner in which Pareto optimality is defined, $P_{current}(t)$ is always a non-empty solution set [19].

$P_{current}(t)$, P_{known}, and P_{true} are sets of MOEA genotypes where each set's phenotypes form a Pareto front (note that Horn [13] uses P_{online}, $P_{offline}$, and P_{actual} instead of the terms $P_{current}(t)$, P_{known}, and P_{true}). We term the associated Pareto front for each of these solution sets as $PF_{current}(t)$, PF_{known}, and PF_{true} (see Figure 2 for an example PF_{true}). Thus, when using an MOEA to solve MOPs, the implicit assumption is that one of the following conditions hold: $P_{known} = P_{true}$, $P_{known} \subset P_{true}$, or that over some norm (Euclidean, RMS, etc.), $PF_{known} \in [PF_{true}, PF_{true} + \epsilon]$.

3. BUILDING BLOCK BACKGROUND

EAs implicitly focus search around building blocks, whose role is defined by the Fundamental Theorem of GAs (or *Schema Theorem*). Goldberg interprets the theorem as meaning "short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations" [10], and refers to these highly fit schemata as *building blocks.* He also postulates the *Building Block Hypothesis*:

> "Short, low-order, and highly fit schemata are sampled, recombined, and resampled to form strings of potentially higher fitness."

Building blocks are an EA's "information source." A building block has two major components: each position has some specified value(s), and each position is somehow dependent upon the others (*linkage*). Building blocks define chromosomes associated with high fitness and in general are used by EAs in three primary ways. First, an initial supply of building blocks is provided via the starting population and used in defining individual fitness. Second, selection should cause an increase in number of the desired building blocks by selecting the individuals containing them for inclusion in the next generation. Finally, building blocks are mixed via recombination; the goal is to assemble the best building blocks into a single individual. In each generation, EAs attempt to exploit the building blocks in their populations subject to selection and recombination.[1]

Few research efforts take steps to *explicitly* incorporate building blocks into EA operation. One that does is Goldberg et al.'s *messy GA* (mGA) [11; 12], a "Top-Down" approach because of the manner in which it attempts to generate appropriate building blocks. This EA is initially of most interest to us because of its explicit enumeration and manipulation of all possible building blocks of a specified size.

3.1 Building Block Observations

We note Goldberg's hypothesis has never been *proved.* Additionally, successful building block use critically depends on the EA representation's degree of *linear separability* (decomposition of the overall problem into subproblems) [17]. If a representation is not linearly separable it suffers from *epistatical* effects (epistasis is a term describing gene interrelationships). Standard EAs can cope with some degree of epistasis, but since exact epistatical relationships are most often unknown, Goldberg's hypothesis may or may not hold true in any given situation.

However, these "negatives" have not prevented successful EA applications based on building blocks. For example, the mGA was used in a practical single-objective application [6]; Deb also implemented a floating point mGA version [5]. When considered at a meta-level, standard EAs (which are predicated upon building block use) often perform much better than random search, implying their use of building blocks (problem domain knowledge) is responsible for their effectiveness and/or efficiency. It then appears that building blocks are useful in *some* problem solving situations. Thus, we focus here on the explicit use of building blocks in solving MOPs.

4. MOPS AND BUILDING BLOCKS

Conjecture 1: Appropriately defined "Building Blocks" can be sampled, recombined, and resampled to form "better" MOP solutions. □

The preceding discussions support the building block concept's practical usefulness. This effort's final goal is determining the validity of Conjecture 1. Although their effectiveness is not yet theoretically quantified, building block concepts can successfully be used in MOEAs [19]. We wish to extend building block concepts successfully applied to single-objective problems into the MOP domain, and use existing analogical concepts in search of more effective and efficient MOEAs.

The Schema Theorem has been historically thought of, described, and analyzed in terms of single-objective functions. However, building block concepts remain applicable when extended to MOPs. We first point out that building blocks are not directly modified by the simultaneous optimization of more than one function. To illustrate, assume a binary-valued genotype of length l that contains several building blocks. Single-objective optimization maps this genotype to

[1] Based upon a mutation probability much less than one, and a building block of any size, it is highly unlikely building blocks will be constructed via mutation. Thus, its effect is usually not considered.

a single value; this is the genotype's associated fitness or phenotype. In MOPs the same genotype maps to a *multi-valued* fitness vector. However, the genotype's structure and its inherent building blocks have not changed in any way! It's simply that two or more fitness functions have been evaluated with respect to this *single* genotype resulting in multiple associated fitnesses.

Single-objective optimization attempts to find a genotype(s) mapping to "high" fitness; MOPs attempt the same. While single-objective optimization problems generally search for a (possibly) unique solution, MOPs often focus on a *set* of effective solutions which may well have very *dissimilar* building blocks! Good MOP building blocks help drive the population towards solutions in P_{true}.

As indicated in Section 2.1, P_{true} defines the MOP's trade-off surface from which some decision maker implicitly indicates acceptable solutions. These solutions may have no clearly apparent relationship besides their membership in P_{true}. In fact, building blocks which are "good" for some Pareto optimal solution(s) may be "not good" for an arbitrarily chosen other (or subset). MOP2 from our proposed MOP test function suite [19; 22] illustrates this:

Minimize $F = (f_1(\vec{x}), f_2(\vec{x}))$, where

$$\begin{aligned} f_1(\vec{x}) &= 1 - \exp\Big(-\sum_{i=1}^{n=2}\big(x_i - \frac{1}{\sqrt{n}}\big)^2\Big), \\ f_2(\vec{x}) &= 1 - \exp\Big(-\sum_{i=1}^{n=2}\big(x_i + \frac{1}{\sqrt{n}}\big)^2\Big)\ , \end{aligned} \qquad (2)$$

where $\vec{x} = (x_1, x_2)$, and $-4 \le x_1, x_2 \le 4$.

Figure 1 shows Equation 2's P_{true} and Figure 2 its PF_{true} (indicated by the '$*$' symbols; dominated vectors are represented by '.'). When analyzing Figure 1 it's easily seen that when taken overall, some relationship (structure) exists between the Pareto optimal solutions in our example. However, a solution in the lower-left hand corner of Figure 1, although closely related to others in its immediate neighborhood, is different from solutions in the upper-right hand corner.[2] Figure 1 graphically identifies building blocks composing solutions in P_{true}. We use these building blocks to gain insight into the Pareto optimal set's nature and (possibly) improve MOEA performance.

4.1 Building Block Importance

It appears that any building block-based search requires a relationship among the genotypes corresponding to (a subset of) Pareto optimal solutions. Note that single-objective EAs already make this assumption and exploit such structure through selection and recombination. They are also able to use building blocks "good" for only some population solutions. This phenomenon is evidenced in successful MOEA approaches predicated on sharing, crowding, and niching techniques [8; 13; 21].

Other MOEA researchers also think this issue significant. We have identified several MOEAs implementing some form of "mating restriction" [19; 21]. Analogous to the term used in computing shared fitness (σ_{share}), a parameter is used describing the "radius" within which solutions must reside in order to recombine. But this is nothing other than what

[2]Figures 1 and 2 are deterministically derived; Pareto representations may slightly change when computational resolution is increased/decreased.

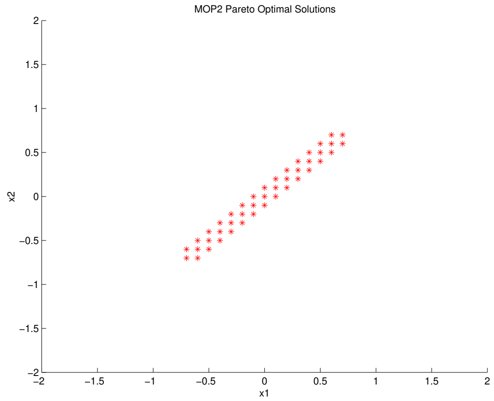

Figure 1: MOP2 P_{true} (genotype)

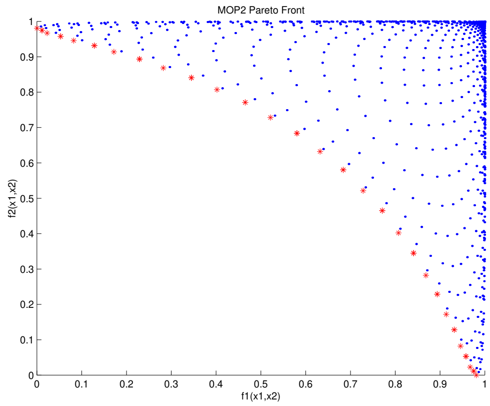

Figure 2: MOP2 PF_{true} (phenotype)

we just stated – building blocks "good" for some solution in P_{known} may be "not good" for others!

Having laid a foundation for understanding building blocks and their use in evolutionary search, we now wish to explore the relationship between building blocks in MOP solutions by extending an existing single-objective building block-based GA to the MOEA domain.

5. THE MULTI-OBJECTIVE MGA

We initially consider the mGA [12] as a vehicle with which to define and investigate MOP building blocks. We select this algorithm for several reasons, although primarily because it was designed to explicitly manipulate appropriate building blocks in order to arrive at an optimal or acceptable solution. The mGA consists of three phases: initialization, primordial, and juxtapositional. At a given computational resolution the mGA initially generates every possible building block corresponding to (a subset of) solutions in P_{true}, and allows determination of which building blocks are present in P_{known}. Each era, during the primordial and juxtapositional phases, evolutionary operators operate upon solutions in a search for "better" fitness.

Additionally, we choose the mGA code as it is freely available, its operation well understood, and its structure modifiable to solve MOPs. This section discusses only the mGA features modified in producing the Multi-Objective mGA (MOMGA). Because of space limitations, we direct the reader to Goldberg et al.'s original papers [11; 12] for a detailed discussion of mGA design and operation.

5.1 Fitness Functions

The mGA requires a fitness function (possibly composed of subfunctions) to be defined over some l-bit string, each $l_i \in \{0,1\}$. The MOMGA currently uses no subfunctions; fitness functions operate on the entire l-bit string. This allows the focus to remain solely on the building blocks used in MOP solutions, and prevents problems in determining the relationship(s) between subfunctions and a complete MOP. Thus, the MOMGA evaluates each of k user-defined fitness functions taking an identical l-bit string as input, performing these actions for each member of the population.

5.2 Fitness Evaluations and Storage

The mGA initially constructs every possible building block of a user-specified size k,[3] resulting in a primordial string population of $2^k\binom{l}{k}$. Each of these strings requires only a *single* function evaluation; the result is stored and used repeatedly during the primordial phase. Likewise in the juxtapositional phase, only one function evaluation is computed per generated individual string, for a total of C evaluations. Thus, the mGA's complexity is $\mathcal{O}(2^k\binom{l}{k}+C)$. The MOMGA requires an increased number of function evaluations, but the increase is linear with the number of simultaneously optimized functions. The number of function evaluations thus increases at a manageable rate, since the complexity of a p-objective MOP is of order $\mathcal{O}(p2^k\binom{l}{k}+pC)$. However, "manageable" is a subjective term. Real-world scientific and engineering MOPs often use computationally complex and time-consuming fitness calculations.

Storage requirements also increase linearly with the number of objectives. The mGA stores a single value from each function evaluation; the MOMGA stores a vector whose number of values corresponds to the number of functions being optimized.

5.3 Selection

The mGA uses tournament selection, which effectively combines selection and scaling into one operation. This is implemented by choosing s individuals at random ($s \geq 2$), and selecting the solution with highest fitness for inclusion in the next generation.

Pareto-based tournament selection (among others) has been successfully used in solving MOPs [13]; comparing vectors based on dominance is a way of finding the "best available" MOP solutions. The PF_{known} presented to a decision maker defines what the "costs" are in respect to all objectives. Thus, we wish to select *currently* effective building blocks/solutions for further processing, and do so by making the MOMGA's tournament selection operator dominance- rather than fitness-based.

The MOMGA randomly selects both two candidate solutions for a tournament along with a comparison set of other individuals. Each of the two candidates is compared (using Pareto dominance) against each comparison set member. If one candidate is nondominated and the other is not (with respect to the comparison set), it is selected for reproduction. If neither or both are dominated then sharing is implemented (see Section 5.5 for more details).

[3] In this section only, the variable k is used in its traditional mGA sense.

mGA modifications extending it for use in MOPs are minimal to this point. However, a crucial component of the mGA's successful operation is the *competitive template*. Extending this concept is not straightforward.

5.4 Competitive Templates

The mGA uses a competitive template in both its primordial and juxtapositional phases. The primordial phase evaluates all building blocks with respect to the template; the juxtapositional phase uses the template to evaluate fitness of the "cut and splice" operator's results. The competitive template separates the value of some bit combination from an entire string without using prior functional knowledge. Each partial string's assigned fitness is then actually a *template fitness*, where the string's unassigned values are filled with the corresponding template values. Template use allows consistent evaluation of partial strings. The mGA uses competitive templates locally optimal to the previous era; a randomly generated template is used to find the locally optimal template for level 1, the resulting "best" answer becomes the locally optimal template for level 2, etc. However, a given template optimizes only one solution with respect to the available building blocks.

At the end of a given level of processing the competitive template is changed to reflect the highest template fitness value achieved, determined by substituting some partial string's values into the current template. However, this competitive template also represents the best *total* solution yet known to the mGA. It is here we arrive at the crux of the matter. The traditional mGA is concerned with finding a *single* ("best") answer. The template limits mGA search and is thus critical in finding the *optimum* and not just an *optimal* solution. As we noted that MOPs deal with a *set* of solutions, our problem is then how to extend the template concept in order to provide the desired set of solutions.

The easy answer, that of using a template for each solution in P_{known}, is not feasible. This implies determining P_{known}'s cardinality *a priori*. Then, how should these templates be employed?

The following strategy is used at present. During its primordial and juxtapositional phases the MOMGA uses a competitive template for *each objective function* being optimized. Each time a partial string's template fitness is computed, a random template is selected from the k available. At the end of each era the "best" answer for each objective replaces the respective current template. We realize there are problems with mGA competitive templates [8; 12]; again note the present intent is determining the use and role of building blocks in forming MOP solutions.

5.5 Sharing

Along with others, Horn [13] has noted that Pareto-based MOEAs require explicit niching and fitness sharing to prevent genetic drift. He and Nafpliotis implement such a scheme in their proposed Niched Pareto Genetic Algorithm (NPGA) [14]. We implement an identical fitness sharing algorithm in the MOMGA.

As described in Section 5.3, two randomly selected candidates are compared (using Pareto dominance) against each solution in a comparison set. If neither or both are dominated, sharing occurs by determining the number of known vectors (the *nichecount*) within some phenotypical niche radius (σ_{share}) of the two candidates. The candidate with the smaller nichecount is then selected. Horn [14] terms this procedure *equivalence class sharing* because these solutions can be considered "equally" fit. Finally, σ_{share} is recomputed each juxtapositional generation using guidelines suggested by Fonseca and Fleming [9].

6. INITIAL EXPERIMENTAL RESULTS

We implemented the MOMGA and generated results for several numeric MOPs as part of a proposed MOEA test suite [19; 22]. In this section we present and explain selected experimental results.

6.1 MOMGA v1.0

Figure 3 presents a diagram of our MOMGA design and implementation. We performed all mGA modifications discussed in Sections 5.1 through 5.5, along with adding and maintaining a *secondary* solution population (P_{known}). In each era, after each juxtapositional generation, $P_{current}(t)$ is added to $P_{known}(t-1)$. At MOMGA termination all vectors of PF_{known} are tested, removing solutions from P_{known} whose associated vectors are dominated. Solution culling is performed at this time so as to not slow MOMGA execution.

6.2 Experimental Methodology

Although numeric test functions provide a common basis for MOEA comparison, these comparisons are still empirical unless PF_{true} is known. For most MOPs of any complexity PF_{true} is unknown, however, there is a way to determine it at a given computational resolution! In a nutshell, exhaustive deterministic evaluation of all possible solutions (represented as l-bit binary strings) provides an MOP's PF_{true} at that resolution. Other algorithms using the same representation and mapping can then be absolutely compared against the MOP's optimum. This methodology, which uses parallel computation on very large-scale platforms, is discussed elsewhere [19; 22].

6.3 Results

A representation size of 24 bits was used for all experiments, resulting in a search space of 2^{24}. For each example, the MOMGA performed fewer than 2^{16} solution fitness evaluations, or less than 0.4% of the search space.

Of initial interest is whether or not the MOMGA appears effective. Figures 4, 5, and 6 show PF_{true} and the MOMGA-derived PF_{known} for three examples. Figure 1 is Equation 2 with three decision variables; the other two are previously identified example MOPs [19]. Figures 1 and 2 reflect minimization MOPs, and Figure 3 a maximization MOP. These examples were selected because PF_{true}'s structure is different for each [19]. In all figures, the '.'s are vectors in PF_{true} and the 'x's are vectors in PF_{known}.

The MOMGA-derived Pareto fronts are in close agreement with those deterministically derived. These results were produced in minutes on a Sun workstation using primarily Goldberg's mGA default parameters. Although pedagogical examples, these results do indicate the MOMGA is identifying and retaining appropriate *and dissimilar* building blocks composing Pareto optimal solutions. We have used the MOMGA in more complex experiments comparing performance of several MOEAs on a selected MOP test suite [19; 21]. With our set of refined experimental metrics [19; 22], initial MOMGA performance is as good as and sometimes better than other published MOEAs.

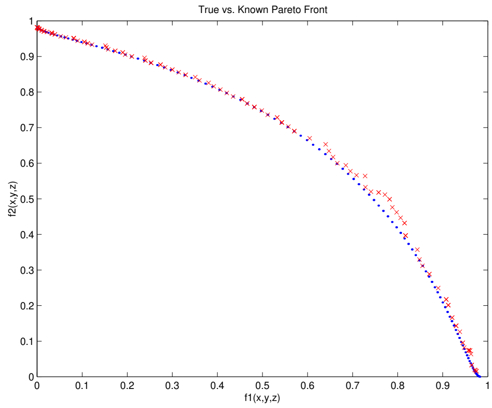

Figure 4: PF_{true} / PF_{known} Comparison

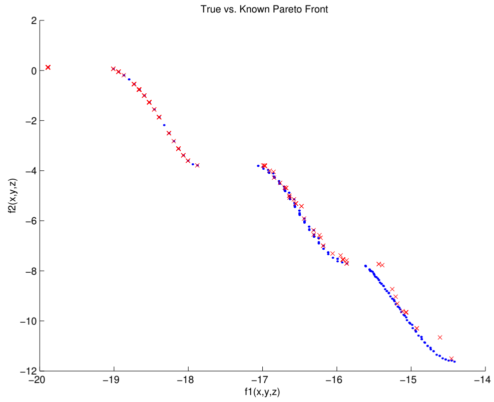

Figure 5: PF_{true} / PF_{known} Comparison

Three other MOEA implementations were selected for testing along with the MOMGA. These algorithms and their original *raison d'être* are:

1. **MOGA**. Implemented by Fonseca and Fleming [9]. Used to explore incorporation of decision maker goals and priorities in the multiobjective search process.

2. **NPGA**. Implemented by Horn et al. [14]. Used to explore benefits of providing P_{known} as input to a Multi Attribute Utility Analysis [16] process.

3. **NSGA**. Implemented by Srinivas and Deb [18]. Used to explore Goldberg's Pareto ranking [10] and prevent bias in exploring the Pareto front.

Appropriate metrics should be selected for judging experimental results concerning MOEA effectiveness (i.e., producing an acceptable result). Thus, specific effectiveness metrics are drawn from those discussed elsewhere [19; 22]. These include error ratio, generational distance, hyperarea ratio, maximum error, overall nondominated vector generation and ratio, and spacing.

Figure 7 presents tested MOEA performance as regards generational distance, a measure of how "far" PF_{known} is from PF_{true}. The MOP1 values for both the NPGA and NSGA were large enough to skew the results when viewed in this format – the graph truncates those two bars (their respective results are $G \approx 66$ and $G \approx 11$). The Kruskal-Wallis

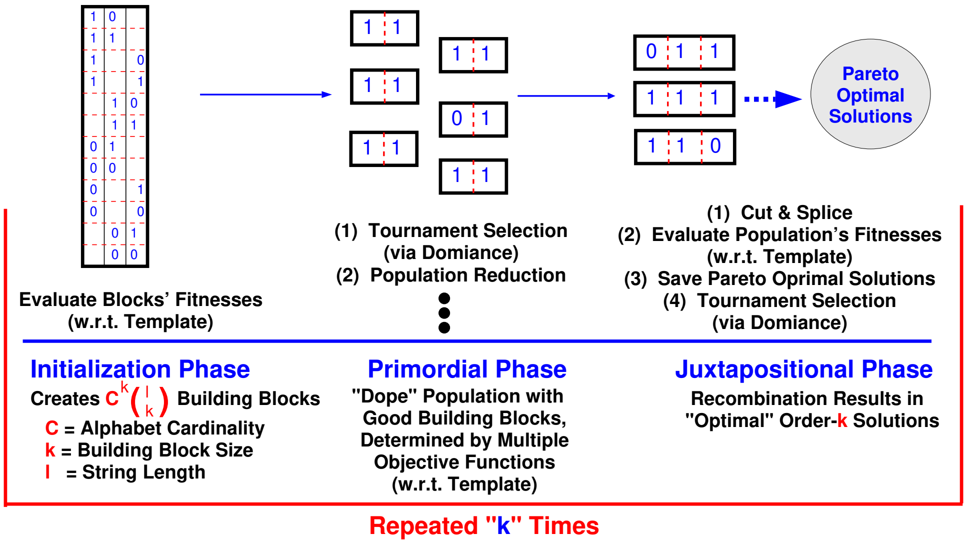

Figure 3: MOMGA Operation

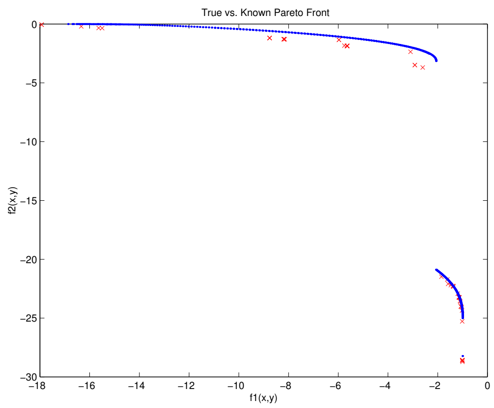

Figure 6: PF_{true} / PF_{known} Comparison

H-Test and pairwise Wilcoxon rank sum tests show many MOEA pairs' results to be statistically insignificant [19]. These and other qualitative results thus allow us to state that in general, when considering generational distance the MOGA, MOMGA, and NPGA gave better results than the NSGA over the test suite problems. This is certainly true for MOP4; the MOGA and MOMGA perform much better than the other two MOEAs on MOP6. Space does not permit the presentation of performance for the other metrics, but their results also statistically support the MOMGA's effectiveness [19].

7. CONCLUSIONS AND FUTURE WORK

We have presented relevant MOP and building block concepts along with a consistent Pareto notation for MOEAs, thus giving a basis for our discussion. The building block concept for single-objective GAs was then applied to MOPs. This extension resulted in the MOMGA, which specifically manipulates building blocks in its search for desired MOP solutions, and was designed and implemented by appropriately modifying the mGA. Experimental results indicate the MOMGA gives excellent performance over our test suite [19]. Even beyond the proposed test suite, we plan to evaluate MOMGA and other MOEA performance when solving more complex MOPs (to include MOPs with extensive side constraints), and apply any insights to real-world MOP solving. Our further research includes examining MOMGA implementation of probabilistically complete initialization and building block filtering, and analyzing both deception in MOPs and how the Pareto front's structure impacts the search process. This analysis attempts to find accurate and quantitative descriptions of building block use in MOPs. We are also developing conjectures and theorems (in terms of efficiency and effectiveness) concerning building block use in MOPs, in order to build on those we present elsewhere [19].

8. REFERENCES

[1] *Proceedings of the 1999 Congress on Evolutionary Computation*. IEEE Service Center, 1999.

[2] T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*, volume 1. IOP Publishing Ltd. and Oxford University Press, 1997.

[3] W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[4] C. A. C. Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[5] K. Deb. *Binary and Floating-Point Function Optimization Using Messy Genetic Algorithms*. PhD thesis, University of Alabama, Tuscaloosa, AL, 1991.

[6] A. Dymek. Examination of Hypercube Implementations of Genetic Algorithms. MS Thesis, AFIT/GCS/ENG/92M-02, EN, WPAFB, Dec 1992.

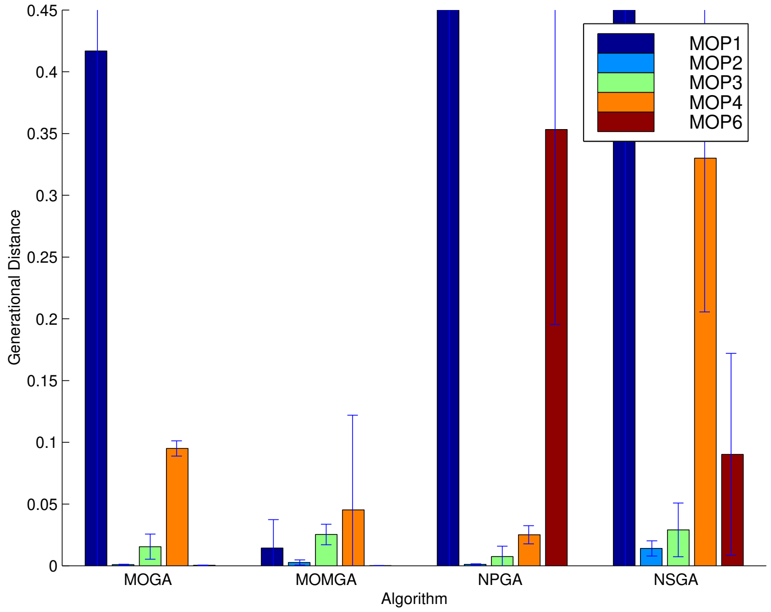

Figure 7: Overall Generational Distance Performance

[7] C. M. Fonseca and P. J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[8] C. M. Fonseca and P. J. Fleming. *Handbook of Evolutionary Computation*, chapter C4: Multiobjective Optimization, pages C4.5:1 – C4.5:9. Volume 1 of Bäck et al. [2], 1997.

[9] C. M. Fonseca and P. J. Fleming. Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms – Part I: A Unified Formulation, and Part II: Application Example. *IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans*, 28(1):26–47, January 1998.

[10] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, Reading, MA, 1989.

[11] D. E. Goldberg, K. Deb, and B. Korb. Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale. *Complex Systems*, 4:415–444, 1990.

[12] D. E. Goldberg, B. Korb, and K. Deb. Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, 3:493–530, 1989.

[13] J. Horn. *Handbook of Evolutionary Computation*, chapter F1: Multicriterion Decision Making, pages F1.9:1 – F1.9:15. Volume 1 of Bäck et al. [2], 1997.

[14] J. Horn and N. Nafpliotis. Multiobjective Optimization Using the Niched Pareto Genetic Algorithm. Technical Report 930005, Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 South Matthews Avenue, Urbana, IL 61801-2996, July 1993.

[15] C.-L. Hwang and A. S. M. Masud. *Multiple Objective Decision Making - Methods and Applications.* Springer Verlag, 1979.

[16] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* John Wiley & Sons, New York, NY, 1976.

[17] N. J. Radcliffe. *Handbook of Evolutionary Computation*, chapter B2: Schema Processing, pages B2.5:1 – B2.5:10. Volume 1 of Bäck et al. [2], 1997.

[18] N. Srinivas and K. Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[19] D. A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* PhD Thesis, AFIT/DS/ENG/99-01, Air Force Institute of Technology, Wright-Patterson AFB, June 1999.

[20] D. A. Van Veldhuizen and G. B. Lamont. Evolutionary Computation and Convergence to a Pareto Front. In J. R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 221–228, Stanford, CA, July 1998. Stanford University Bookstore.

[21] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Air Force Institute of Technology, 1998.

[22] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective Evolutionary Algorithm Test Suites. In *Proceedings of the 1999 ACM Symposium on Applied Computing.* ACM, 1999.