SANDIA REPORT

SAND2016-1376
Unlimited Release
Printed February 2016

Holistic Portfolio Optimization using Directed Mutations

Stephen M. Henry, Mark Andrew Smith, John P. Eddy

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: http://www.ntis.gov/search

SAND2016-1376
Unlimited Release
Printed February 2016

Holistic Portfolio Optimization using Directed Mutations

Stephen M. Henry, Mark Andrew Smith, John P. Eddy
System Readiness and Sustainment Technologies
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1188

Abstract

Genetic algorithms provide attractive options for performing nonlinear multiobjective combinatorial design optimization, and they have proven very useful for optimizing individual systems. However, conventional genetic algorithms fall short when performing holistic portfolio optimizations in which the decision variables also include the integer counts of multiple system types over multiple time periods. When objective functions are formulated as analytic functions, we can formally differentiate with respect to system counts and use the resulting gradient information to generate favorable mutations in the count variables. We apply several variations on this basic idea to an idealized "hanging chain" example to obtain ≫1000x speedups over conventional genetic algorithms in both single- and multiobjective cases. We develop a more complex example of a notional military portfolio that includes combinatorial design variables and dependency constraints between the design choices. In this case, our initial results are mixed, but many variations are still open to further research.

ACKNOWLEDGMENTS

We are indebted to the LDRD Program and especially the Defense Systems and Assessments Investment Area for funding this research which has allowed us to pursue some of our intellectual interests while developing capabilities that will help us in our future national security engagements. Specifically, we would like to thank Jim Hudgens, Terry Stalker, and the rest of the Precision Engagement Investment Area Team for listening to our ideas, asking challenging questions, and alerting us to related happenings across Sandia. Thanks as well to their staff and those in the LDRD office for keeping us on schedule. Special thanks to Alan Nanco, our Program Manager and the Precision Decisions POC, for his guidance and support from before our initial idea submission through to this day. Thanks to our manager, Bruce Thompson for encouraging us to think of LDRD ideas, helping with transition efforts, and for being supportive in so many other ways. Finally, we would like to thank Joyce Gearou, Alice Karé, and Zachary De Gregorio in the Energy Technology & System Solutions business operations team. They provided the cost estimates, spend plans, budget trackers, and landing paths which made a breeze of the financial aspects of proposing, setting up, managing, and finishing this project.

CONTENTS

FIGURES

TABLES

NOMENCLATURE

CONOPS	Concept of Operations
CPAT	Capability Portfolio Analysis Tool
EA	Evolutionary Algorithm
GA	Genetic Algorithm
GD	Gradient Descent
HPO	Holistic Portfolio Optimization
JEGA	John Eddy Genetic Algorithm
MECE	Minimum Elastic Catenary Energy
MILP	Mixed Integer Linear Programming
PO	Portfolio Optimization
SDO	System Design Optimization
UAV	Unmanned Aerial Vehicle
WSTAT	Whole Systems Trade Analysis Tool

1 INTRODUCTION

In the field of optimization, evolutionary algorithms (EAs) and gradient descent (GD) techniques represent two very important yet seemingly unassociated methodologies [1]. Speaking broadly, evolutionary algorithms iteratively generate *populations* of solutions that change over time via operations akin to genetic mutation (subtle changes in the solution properties) and crossover (combining properties of parent solutions to create new children). These populations evolve over time guided by a "survival of the fittest" criterion (often Pareto dominance) that propagates good solutions to subsequent generations while dropping bad solutions. Gradient decent algorithms, on the other hand, generally start with a *single* solution that is iteratively updated by "moving" some distance in a direction suggested by the derivative of the objective function(s) – i.e., always moving "downhill" from the current solution location in a minimization problem.

Not only are the mechanics of these two approaches quite distinct, but so too are the types of problems to which they are typically applied. Evolutionary algorithms are generally favored when seeking *near globally optimal* solutions to a problem with *many competing* objective functions which may be *non-differentiable*. Gradient descent methods are customarily used to find a *locally optimal* solution to a *single* objective function that must be *differentiable*. EAs can be thought of as *zeroth-ordered* techniques in that they require no information about directions of improvement from objective function gradients; they instead discover these directions by modifying, combining, and propagating the genes of improving solution population members. GD approaches are *first-order* techniques that explicitly require gradient information to obtain directions of improvement. Because of this, EAs are typically able to escape from local minima and eventually converge towards the overall globally optimal solution(s) whereas GD methods are normally trapped in the nearest local minima. The tradeoff, however, is that EAs only "drift" towards optimal solutions while GD algorithms can quickly "hone in" on the local optimal by directly following gradients.

The goal of this research is to solve a very challenging and previously intractable type of optimization problem which we refer to as Holistic Portfolio Optimization (HPO) – a type of optimization that combines elements of optimal system design and optimal portfolio composition together within a single problem. In order to solve these types of problems, we develop a new framework that captures the desirable aspects of both EAs and GD methods simultaneously by using first-order gradient information where possible as an additional means of updating an EA solution population – an approach we refer to as Directed Mutation. It is important to note that we do not replace the standard genetic operations of mutation and crossover; instead we employ Directed Mutation in synergy with these operations in order to dramatically increase the speed of EA convergence towards globally optimal solutions. To complicate matters, the HPO problems we intend to solve usually involve large numbers of *integer* variables (specifically, the number of systems in a portfolio at a certain time). Thus, the gradient information leveraged by Directed Mutation must be able to not only provide directions of improvement, but also preserve integrality of the decision variables that are mutated. While other works have studied similar ideas around the combination of gradient information and evolutionary algorithms (often focused on EAs with real-valued variables and a single objective function [2, 3, 4, 5, 6, 7]), this research is unique in its application of these techniques to both 1) integer-valued decision variables and 2) multiple competing objective functions.

The remainder of this report is laid out as follows. Section 2 provides a more thorough description of holistic portfolio optimization problems and how they compare to more traditional

system design optimization and portfolio optimization. Section 3 covers the implementation of Directed Mutation within a genetic algorithm framework known as JEGA. Section 4 introduces an HPO problem surrogate called the Minimum Elastic Catenary Energy (MECE) problem – a relatively simple optimization problem with intuitive optimal solutions and gradient interpretations that provides a testbed for Directed Mutation approaches. This section discusses various methods by which objective function gradients can be incorporated and aggregated into both single and multi-objective MECE solvers and demonstrates impressive speedups over a standard GA. Section 5 introduces a notional HPO problem that involves optimal city-scale counterinsurgency operations using a portfolio of unmanned aerial vehicles (UAVs), ground vehicles, and soldiers. Attempts at applying the techniques that worked so well on the MECE have so far been inconclusive when applied to this HPO: some speedups are observed early in the evolution, but it remains an open question as to why these speedups are lost as the evolution progresses. Finally, Section 6 summarizes our work and provides suggestions for continued investigation.

2 MOTIVATION

The Directed Mutation methodology described in this report was developed to enable Holistic Portfolio Optimization – the ability to optimize not only the *composition* of a portfolio through time, but also the *design* of individual items within the portfolio in order to achieve more desirable overall behavior. This type of problem is extremely challenging and is typically not tractable at scale with current optimization methods. A large scale portfolio may contain hundreds of possible "systems" (we use this term generically to mean any item within a portfolio which itself may have many possible configurations, such as a truck, a project, or even a person). Each system within the portfolio may contain dozens of component parts each with dozens of possible realizations (for instance, a truck configuration may choose from among several possible engines, drivetrains, chassis, etc.). The task of optimizing the integer numbers of systems in the portfolio across many time periods (typically 10 to 30) and simultaneously optimizing the design choices of the systems induces an astronomically huge search space – one that cannot be addressed by current methods.

Because of this intractability, it is standard practice to separate the optimization of individual systems from the optimization of the overall portfolio composition. This separation limits the portfolio optimization to fixed system configurations; it is blind to the full trade space of design possibilities – some of which might be preferable. This separation also implies that the system design optimization is blind to broader portfolio considerations such as ideal price or timing of system availability relative to other systems. Because these separate problems are in reality highly interdependent, preselecting a fixed system design (even a Pareto optimal design that balances competing design objectives) for incorporation into the portfolio optimization can create highly suboptimal solutions from a holistic, unseparated perspective.

A real-world example of the deleterious effects of disjoint system design and portfolio optimization can be seen in the development and ultimate cancellation of the U.S. Army's Ground Combat Vehicle (GCV) program – a large-scale acquisition program designed to replace the Bradley Infantry Fight Vehicle. During initial GCV development, two separate tools were designed by Sandia National Labs to address the system design optimization and portfolio optimization challenges faced by the Army's ground combat fleet – the Whole System Trades Analysis Tool (WSTAT) and the Capability Performance Analysis Tool (CPAT), respectively. Though GCV designs were heavily informed by WSTAT and the fleet modernization plan was continually informed by CPAT, the ultimate GCV design did not conform to the overall budget and performance needs of the entire fleet – especially after the introduction of severe sequestration budget cuts. This separation of system design and portfolio optimizations led to case studies where the CPAT fleet optimization determined it was optimal to *avoid fielding* GCV, even though the GCV configurations was a Pareto optimal solution from the WSTAT trade space. This was not a problem of incompatible tools, but rather a symptom of the disjoint system design and portfolio optimizations. In essence, the GCV design was a good tradeoff from the standpoint of the *individual system*, but it was a poor tradeoff from the standpoint of the *entire fleet*. Thus, in February 2014 the GCV program was terminated – having already spent over $1 billion.

Examples such as this motivate our use of Directed Mutations as an enabling framework for HPO. To lay a foundation for subsequent discussion, the remainder of this section presents a more formal description of system design optimization, portfolio optimization, and holistic portfolio optimization – comparing and contrasting the elements of each and motivating the

rationale for Directed Mutations as a promising approach for speeding up evolutionary algorithms when applied to HPO.

2.1 System Design Optimization

The goal of system design optimization (SDO) is to discover configurations for a single system that optimally balance many competing design objectives (such as performance, cost, risk, design margin, etc.). Rather than producing a single "best" solution, SDO presents decision makers with many different solutions that each balance the competing objectives in unique ways (a set of system designs that is referred to as the Pareto optimal trade space). SDO is predicated on the ability to decompose a system into component subsystems where each subsystem has multiple potential design choices (or parts) that could be used. SDO explores this combination of subsystem choices (creating specific system configurations) in order to find the best designs in multiple competing objectives [12]. A general formulation for SDO is as follows:

$$\begin{aligned} \text{SDO: } & \max \ \tilde{f}_1(\boldsymbol{x}),\ \tilde{f}_2(\boldsymbol{x}),\ \dots,\ \tilde{f}_n(\boldsymbol{x}) \\ & \text{s.t. } \boldsymbol{x} \in \widetilde{\Omega} \\ & \quad x_i \in \{design_{i,1}, \dots, design_{i,n_i}\}. \end{aligned}$$

Here, the vector of design decisions $\boldsymbol{x}$ represents the decomposition of the system into subsystems, and each discrete variable x_i determines the design selected for subsystem i. The objective functions $\tilde{f}_1(\boldsymbol{x})$ through $\tilde{f}_n(\boldsymbol{x})$ can be general, nonlinear measures evaluated based on the selection of subsystem designs (here and throughout this paper we use the $\widetilde{\blacksquare}$ notation to indicated a nonlinear function or system of constraints). The constraints $\boldsymbol{x} \in \widetilde{\Omega}$ represent a wide variety of nonlinear restrictions including subsystem design necessitations or obviations (i.e., "Engine 1" is not allowed to be selected with "Drivetrain 3").

Due to the nonlinear, multi-objective nature of SDO, these problems are often solved via evolutionary algorithms. When the number of objective functions is small (generally on the order of 5 or 6) and there are relatively few constraints (less than a few dozen), classical EA approaches such as genetic algorithms achieve good results.

2.2 Portfolio Optimization

In contrast to SDO, the goal of portfolio optimization (PO) is to find the single best "modernization plan" that schedules *which* systems should be included in the portfolio, *how many* should be included, and *when* they should be incorporated [13]. In contrast to SDO which employs design variables to represent systems, PO uses fixed system designs and schedules the inclusion or exclusion of these fixed designs through time to find the single best solution that optimizes the portfolio design goal (such as maximize performance or minimize risk). Often when SDO and PO are performed independently, the portfolio optimization occurs *subsequent to* the system design optimization – incorporating as inputs the fixed properties of the system configuration(s) chosen by decision makers from each SDO trade space. The portfolio schedules are typically subject to numerous business rules that constrain the allowable behavior of the plan (such as budgets, production limits, etc.). A general PO problem can be formulated as follows:

$$\begin{aligned} \text{PO: max} \quad & \bar{F}(\boldsymbol{y}) \\ \text{s.t.} \quad & \boldsymbol{y} \in \bar{\Omega} \\ & y_{j,t} \in \mathbb{Z}_+ \quad \forall j \in \{1, \dots, S\}, t \in \{1, \dots, T\}. \end{aligned}$$

Here, the integer decision variable $y_{j,t}$ indicates how many systems of type j should be in the portfolio at time t. Because 1) there may be many systems, 2) the integer number of systems may be quite large, and 3) the number of business rules may also be very large, the typical solution methodology for PO is based on Mixed Integer Linear Programming (MILP), which employs a tree-based search strategy to intelligently enumerate and explore candidate solutions to produce a global optimal solution (or a near optimal solution with gap information). MILP techniques are very mature optimization approaches and can solve extremely large-scale problems with tens-of-thousands of integer variables and hundreds-of-thousands of constraints. The restriction, however, is that only a single linear objective function, $\bar{F}(\boldsymbol{y})$, can be specified and all constraints, $\boldsymbol{y} \in \bar{\Omega}$, must be linear (we use the $\bar{\blacksquare}$ notation throughout this paper to indicate a linear function or set of constraints).

2.3 Holistic Portfolio Optimization

As previously mentioned, the goal of holistic portfolio optimization is to combine the SDO and PO problems into a unified optimization – simultaneously answering 1) what the design of individual systems should look like, 2) which of these systems should be integrated into the portfolio, 3) how many should be included, 4) when they should be incorporated, and 5) how the system designs should evolve over time. In addition, HPO must address multiple competing portfolio-level goals simultaneously – finding a Pareto optimal trade space of portfolio plans that balance these objectives (such as portfolio performance, portfolio cost, portfolio risk, etc.). Note that in contrast to the trade space of a single system produced by SDO, the HPO trade space is a population of *comprehensive portfolio plans* – each plan incorporating multiple systems (each with multiple subsystems) and balancing the portfolio-level objectives in a unique way. A general formulation for HPO is shown below, combining elements from the SDO and PO formulations:

$$\begin{aligned} \text{HPO: max} \quad & \tilde{F}_1(\boldsymbol{z}), \tilde{F}_2(\boldsymbol{z}), \dots, \tilde{F}_N(\boldsymbol{z}) \\ \text{s.t.} \quad & \boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y}) \\ & \boldsymbol{x} \in \widetilde{\Omega} \\ & \boldsymbol{y} \in \bar{\Omega} \\ & \boldsymbol{z} \in \widetilde{\Phi} \\ & x_{i,j,t} \in \{design_{i,1}, \dots, design_{i,n_i}\} \quad \forall j \in \{1, \dots, S\}, t \in \{1, \dots, T\} \\ & y_{j,t} \in \mathbb{Z}_+ \quad \forall j \in \{1, \dots, S\}, t \in \{1, \dots, T\}. \end{aligned}$$

Here the decision variables $\boldsymbol{x}$ and $\boldsymbol{y}$ serve similar roles as they do in SDO and PO, respectively, where $x_{i,j,t}$ gives the design selection of subsystem i for system j at time t and $y_{j,t}$ gives the integer number of systems j in the portfolio at time t. The nonlinear system design constraints $\boldsymbol{x} \in \widetilde{\Omega}$ and the linear portfolio business rules $\boldsymbol{y} \in \bar{\Omega}$ are also incorporated from SDO and PO, along with a new set of general nonlinear restrictions $\boldsymbol{z} \in \widetilde{\Phi}$ which co-constrain system design and system count decisions (e.g., the number of systems of type j might be limited by

tight production constraints if Engine 1 is chosen for subsystem i). Finally, note that the potentially nonlinear objectives $\tilde{F}_1(\boldsymbol{z})$ through $\tilde{F}_N(\boldsymbol{z})$ are functions of both system counts *and* system design decisions.

This HPO problem is extremely challenging as it combines aspects of multiple simultaneous SDOs and one PO – each of which are suited to very different solution approaches. The large integer variables $y_{j,t}$ and the considerable number of linear business rules $\boldsymbol{y} \in \overline{\Omega}$ are well suited for MILP approaches. However, the multiple nonlinear objective functions, the nonlinear design restrictions $\boldsymbol{x} \in \widetilde{\Omega}$, and design/count co-restrictions $\boldsymbol{z} \in \widetilde{\Phi}$ preclude MILP techniques. On the other hand, while HPO's nonlinear and multi-objective nature suggests an EA approach, standard EAs will have great difficulty solving over 1) the large integer variables $y_{j,t}$ and 2) the large numbers of overall constraints. A great deal of literature is dedicated to the study of highly constrained evolutionary optimization [8, 9, 10, 11], however little attention is given to the difficulties of large integer decision variables – especially over multiple objective functions. Hence, it is this first difficulty (evolution over large integer decision variables) that we focus on in this research via the use of Directed Mutation – guiding evolution over the large integer decision variables to greatly speed population convergence towards Pareto optimal trade spaces.

3 DIRECTED MUTATION IMPLEMENTATION

To test the idea that gradient-guided mutations improve evolutionary convergence for HPO problems, we use the John Eddy Genetic Algorithm (JEGA) [14] as our EA framework. JEGA provides several advantages in that 1) the source code is readily available, 2) the algorithm is computationally efficient and had been matured over many years, and 3) it has already been implemented within Sandia optimization tools for system design and portfolio optimization (Whole System Trades Analysis Tool [WSTAT] [12] and Technology Management Optimization [TMO] [15, 16], respectively).

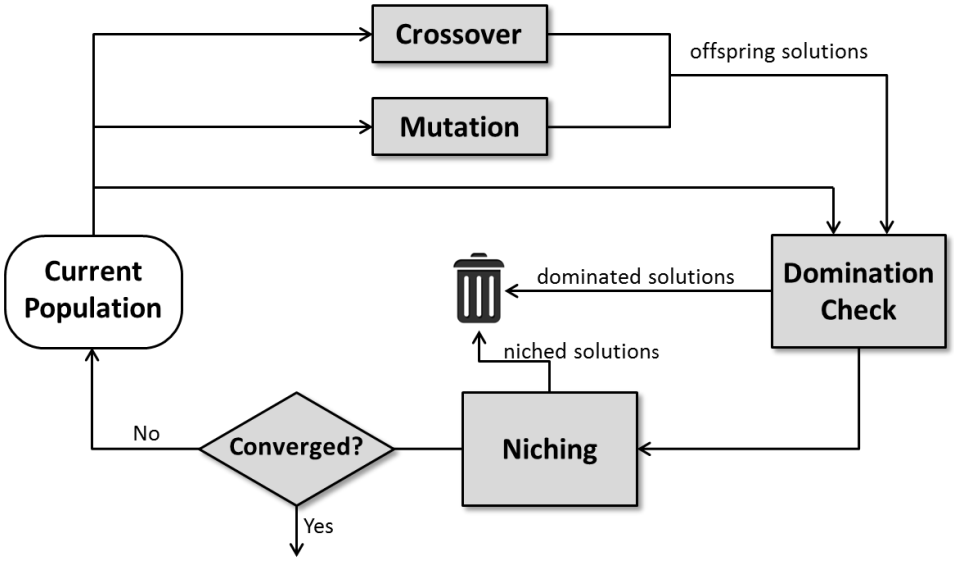


Figure 1. Standard JEGA algorithm flow.

To illustrate our implementation of Directed Mutations, we first give a brief overview of the baseline JEGA algorithm using Figure 1 as a guide and in the context of SDO optimization. Starting with the current population of solutions (typically the initial population is randomly generated), population members are passed to both the Crossover and Mutation operators. Crossover combines properties of randomly selected "parent" solutions together to make new "child" solutions. For example, suppose that the current population has Parent 1 = (Engine 1, Drivetrain 1, Chassis 1, Brake 1) and Parent 2 = (Engine 2, Drivetrain 2, Chassis 2, Brake 2). The Crossover operator might create Child = (Engine 2, Drivetrain 2, Chassis 1, Brake 1) by combining the first part of Parent 2's "gene" with the second part of Parent 1's gene. The number of child solutions, the number of breakpoints (where parent genes are joined together) and even the number of parents are all JEGA parameters that can be set for the Crossover operator.

Whereas Crossover recombines the *already existing genes* of current population members, the Mutation operator modifies the genes of population members, thus creating *new genetic material* by altering a randomly selected design variable. For example, when the solution (Engine 1, Drivetrain 1, Chassis 1, Brake 1) enters the Mutation operator, it might be changed to (Engine 3, Drivetrain 1, Chassis 1, Brake 1). The number of mutated solutions and the manner of mutation can be set by parameters for the Mutation operator.

Next, the offspring solutions from Crossover and Mutation, along with the original solutions from the current population, are sent to a Domination Check operator. When comparing two

solutions, we say that one solution is "dominated" by the other if it is as bad as or worse than the other solution in all objective functions. The Domination Check operator uses this Pareto dominance condition and serves as a survival-of-the-fittest mechanism to cull some or all of the dominated solutions (some dominated solutions may be retained in order to preserve a minimum number of solutions or to avoid overly-rapid shrinkage of the population).

The surviving solutions are then sent to the Niching operator, whose task is to further cull the solutions and provide a sparse, evenly-spaced representation of the current population. This is a necessary procedure due to the usually enormous size of the true discrete Pareto population; even for a modest SDO problem this full population would not fit within computer memory and so needs to be represented in a sparse yet representative manner.

Finally, the Convergence operator compares the population output from Niching with populations from a number of previous generations – tracking changes in the breadth, progress, and density of the populations and terminating the algorithm if solution populations are no longer changing or improving (or if other limits such as run time or number of generations is met). Otherwise, the niched population becomes the new Current Population and the cycle repeats until termination criteria are achieved.

Figure 2 demonstrates how we incorporate the new Directed Mutation operator within the traditional JEGA algorithm loop. As previously mentioned, our desire is not to replace traditional genetic operations: Directed Mutation should work *in synergy* with Crossover and Mutation. With this goal in mind, we implement Directed Mutation as a supplement to the other two genetic operators. Directed Mutation operates on the *offspring solutions* emerging from Crossover and Mutation. In theory, this arrangement allows improvements found by Mutation and Crossover to be *immediately improved upon* using gradient information – speeding up algorithm convergence. Note that both the original offspring solutions (unaltered by Directed Mutation) *and* the gradient mutated solutions (as well as the current population) are all sent to the Domination Check operator. This ensures that all solutions (current generations, mutated solutions, crossover children, and gradient mutated solutions) all compete on equal footing for propagation to subsequent generations.

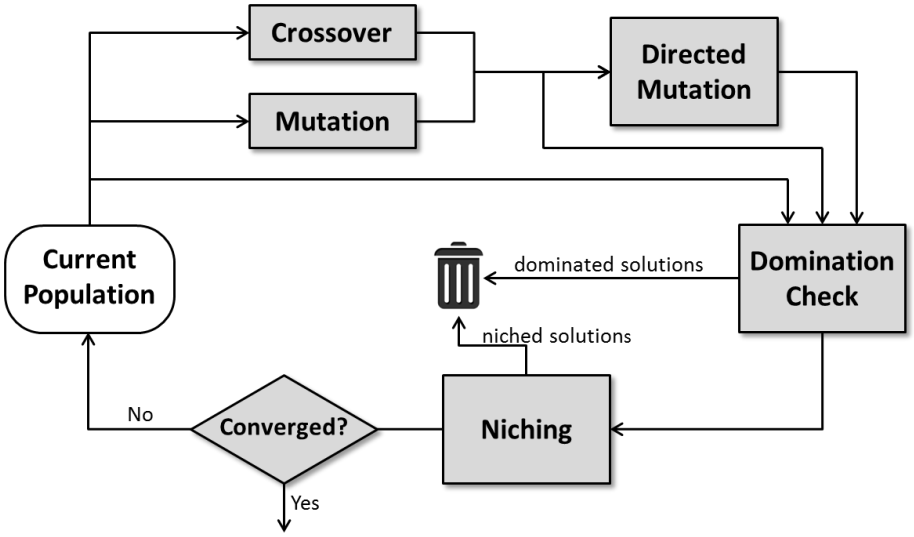


Figure 2. JEGA algorithm flow augmented with directed mutation.

As with the other operators, Directed Mutation is implemented as a modular procedure within the flow of the overall genetic algorithm – allowing many potential gradient-guided schemes to be quickly built, tested, and refined. The computational experiments that follow in the next sections investigate a broad variety of gradient-guided techniques. All tests, however, fall within the general framework outlined in Figure 2: only the specific methodology within the Directed Mutation operator varies.

4 HANGING CHAIN PROBLEM

Before designing, formulating, and testing the value of Directed Mutations on HPO problems, we first examine the effect of gradient guidance on a simpler optimization. This surrogate for HPO problems is informally referred to as the "hanging chain" problem, which in the context of physics and geometry, is the equilibrium orientation that a chain acquires when suspended at both ends and drooping under the force of gravity. This equilibrium shape (also known as a catenary) brings into balance the forces of gravity and tension at each point along the chain and assumes an appearance similar to a parabola, though it is actually modeled by a hyperbolic cosine. In the context of optimization, the hanging chain problem can be thought of as the orientation in space that *minimizes* the total potential energy of the system (gravitational potential energy).

This hanging chain problem serves as an excellent HPO surrogate for several reasons. First, the optimal solution is intuitive. We commonly encounter real-world catenaries in such examples as electrical power lines, simple suspension bridges, and rope barriers. Second, the optimal orientation of a hanging chain can be solved directly (and very efficiently) via the hyperbolic cosine. Thus, the convergence of an evolutionary algorithm towards the true optimal can be precisely and accurately quantified – allowing excellent computational comparison of various Directed Mutation schemes. Third, if the hanging chain problem is formulated in "segments" (i.e., the chain is discretized into separate pieces wherein the optimization must find the optimal spatial coordinates for the links between each piece), then genetic algorithms can have great difficulty converging to the optimal solution. This difficulty is due not only to the size of the search space (which can be very large if each link's spatial coordinates can realize a broad range of values), but also due to the degree of colocation required in the independent decision variables of adjoining segments (more on this in the following subsection).

Finally, if the spatial coordinates of each segment are restricted to take on integer values, then the hanging chain mirrors HPO even more closely in that the *adjoining spatial positions* of each segment mimics the *adjoining temporal counts* of the number of systems in the portfolio. That is, just as the optimal hanging chain has a "smooth" shape through space, the number of systems in an optimal portfolio plan often has a "smooth" shape through time (systems are not arbitrarily purchased, divested, and purchased again – incurring unnecessary costs).

4.1 Single Objective Minimum Elastic Catenary Energy Problem

As a formal example of a hanging chain problem, we introduce the Minimum Elastic Catenary Energy (MECE) problem, which consists of n links connected end-to-end by springs and hanging from both ends under gravity. The goal of MECE is to minimize a single objective function – the potential energy of the system – by optimally locating the integer x and y-coordinates, (x_i, y_i), of the links connecting springs i and $i+1$. The MECE problem can be formulated as follows:

$$
\begin{aligned}
\text{MECE: } \min \quad & \tilde{f}(\boldsymbol{x}, \boldsymbol{y}) \\
\text{s.t. } & \tilde{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n-1} p_i + \sum_{i=1}^{n} y_i \\
& p_i = \left(\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} - R \right)^2 \\
& 0 \le x_i \le L \\
& 0 \le y_i \le H \\
& x_i, y_i \in \mathbb{Z},
\end{aligned}
$$

where the nonlinear objective function $\tilde{f}(\boldsymbol{x}, \boldsymbol{y})$ gives the total potential energy of the hanging chain, which is a sum of the spring compression or stretching potential energy plus the gravitational potential energy (i.e., the height, y_i, of each link). The decision variables x_i and y_i give the Cartesian coordinates of the i^{th} link, and these positions are restricted to lie on the integer grid. The constants L, H, and R give the distance between the two supporting posts, the height of these posts, and the rest length of each spring in the system, respectively. The spring compression/stretching potential energy, p_i, is calculated via Hooke's law (i.e. proportional to the square of the spring deformation distance from rest length) where the spring constant is 2 for simplicity. Lastly, the fixed positions $(x_0, y_0) = (0, H)$ and $(x_n, y_n) = (L, H)$ define where the spring system attaches to the left and right post, respectively. Figure 3 shows a near-optimal MECE system with 9 springs and 8 link locations.

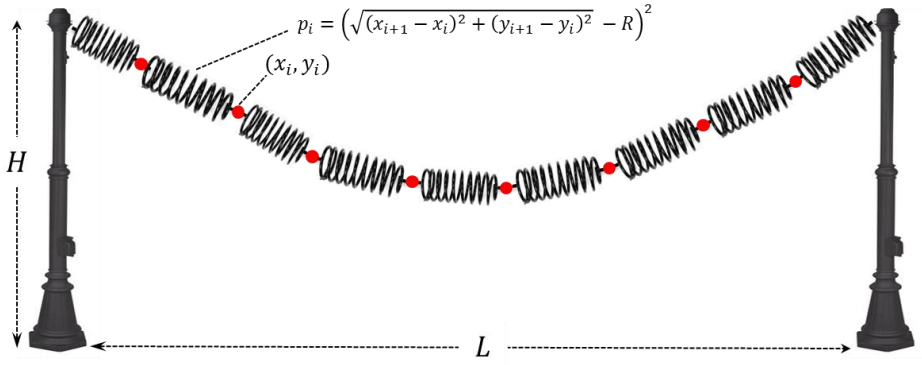


Figure 3. A MECE system of 9 springs hanging under gravity and supported at each end.

As previously mentioned, genetic algorithms can have difficulty converging to an optimal MECE solution due to 1) problem size and 2) variable coordination. To see the first difficulty, consider Figure 3 where $L = 200$ and $H = 300$. Under this modestly granular discretization; each of the eight x-variables can take on 201 possible values while each of the eight y-variables can take on 301 values, yielding a search space of $201^8 * 301^8 \approx 1.8 \times 10^{38}$. The general search space size of $(LH)^n$ can be astronomical even with modest L, H, and n.

The second difficulty, variable coordination, is perhaps more subtle yet more pernicious – arising from both the nature of Mutation and Crossover as well as the implicit structure of MECE. To gain intuition into this difficulty, consider Figure 4 which presents a suboptimal MECE solution both before and after a single mutation operation on the y-coordinate of the third link. Note that even though the mutation brings the third link closer to its globally optimum

location, the post-mutation system actually has *more* potential energy than the original due to increased stretching of the springs attached to the mutated link. What might be more helpful is a coordinated update of links 2 through 6 that moves *all five links up simultaneously* towards their optimal location, but the probability is exceedingly remote that five independent mutations would all act 1) on the same solution, 2) on the proper links, and 3) in the proper direction. Crossover, on the other hand, is able to update many links simultaneously by joining part of one solution with part of another, but recall that Crossover does not create new genetic material. Rather, Crossover recombines *existing* genetic material in new ways. Thus, the expectation that it would be able to generate coordinated improvements to solutions presupposes that the coordinated genes *already exist* in the solution population.

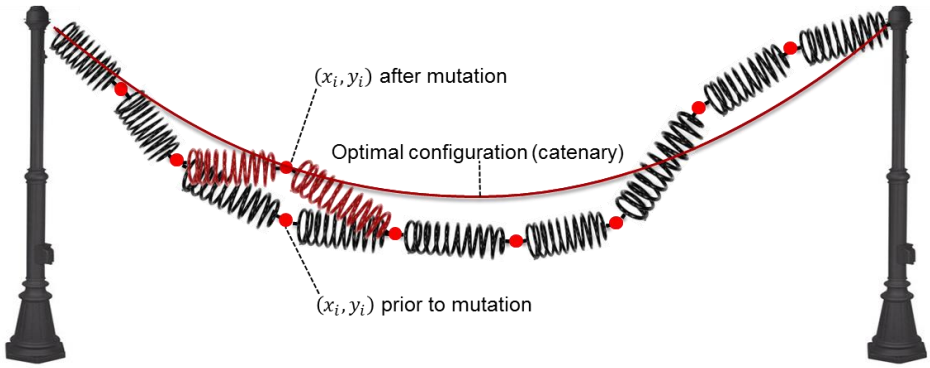


Figure 4. Without coordination, single mutations often increase total potential energy.

These same difficulties are also present to an even higher degree in HPO problems. Tracking the composition of the portfolio through time requires integer decision variables for each system type at each time period – yielding a combinatorially explosive search space. In addition, mutations to an HPO schedule increase or decrease the system count of an individual system in a single time period, thus causing the same types of problem as outlined in Figure 4. Transitioning from a current portfolio plan to a new, improved plan typically involves simultaneous, *coordinated changes across many temporally adjacent time periods*. It is for these reasons that we employ MECE as a simpler yet still challenging surrogate for HPO. Figure 5 demonstrates these computational challenges in an MECE system with $n = 30$ by plotting the best solutions from generations 100, 1000, 100,000, and 1,000,000. Notice that there is significant improvement during the first 1000 generations, but convergence towards the optimal (dotted line) configuration is greatly stifled beyond generation 10,000 (almost no progress is made between 100,000 and 1,000,000). In order to realize improvements beyond the generation 1,000,000 solution, a great number of coordinated changes must occur (for example, the entire left side of the chain system must slide further to the left), and under mutation these changes are astronomically improbable.

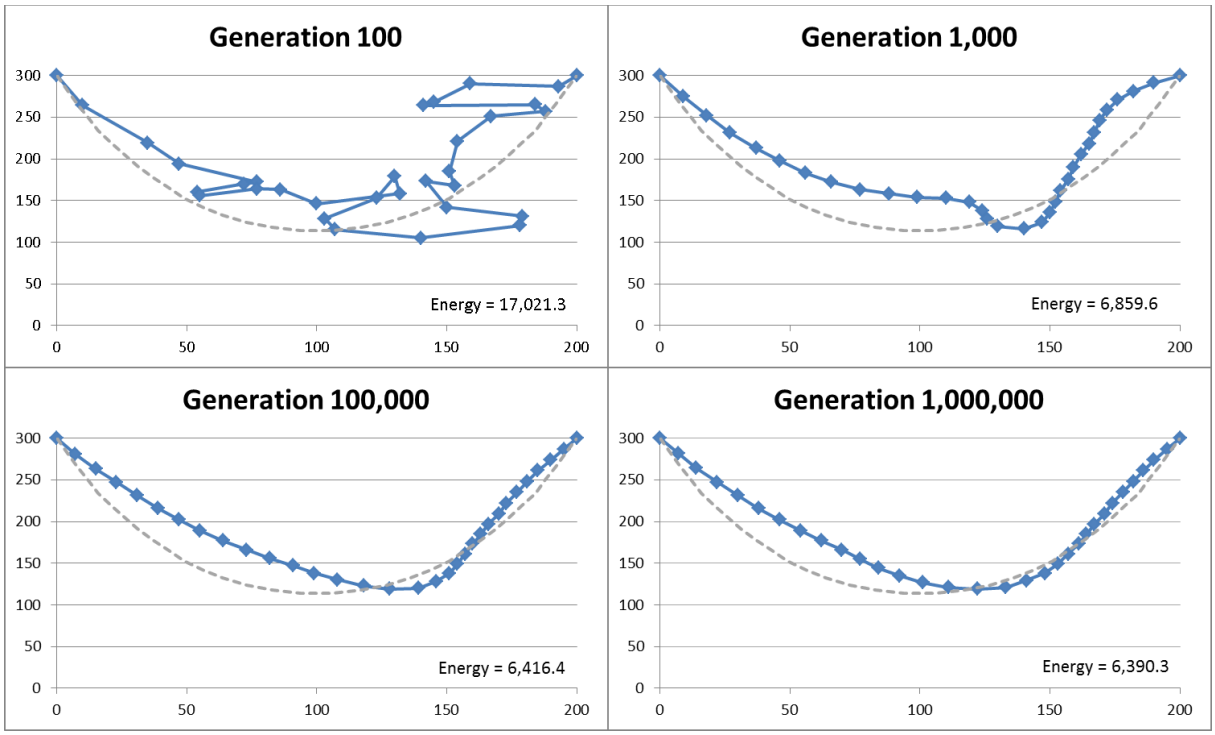


Figure 5. JEGA convergence of an MECE system towards the optimal solution.

These difficulties are what motivate our desire for gradient coordination of EA mutations, and it is another advantage of the MECE problem that gradient information at each link is readily computable and has an intuitive interpretation – namely, the gradient is the negative of the net force vector exerted on that link. Conveniently, the principle of locality provides that the force on each link is only dependent on gravity (a constant in the negative y-direction) and the two springs connected to that link as depicted in Figure 6 – no other parts of the system need to be considered. (This locality property may or may not be present in a full HPO problem and depends on the nature of the underlying value model.) The force exerted by the two springs is calculated via Hooke's law, which applies a force along the coil axis proportional to the deformation of the spring from rest length (pushing if the spring is compressed and pulling if the spring is stretched). In other words, $\vec{F}_{Net_i} = \vec{F}_{Spring_i} + \vec{F}_{Spring_{i+1}} + \vec{F}_{Gravity} = -\left(\frac{\partial \tilde{f}}{\partial x_i}, \frac{\partial \tilde{f}}{\partial y_i}\right)$.

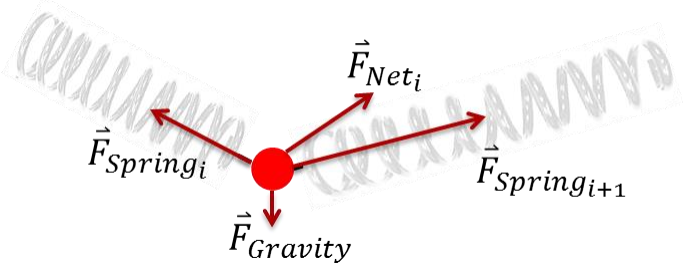


Figure 6. The MECE gradient at each link is the negative of the net force on that link.

This simple force derivation at each link provides a powerful means by which to update the hanging spring system (i.e. Directed Mutations). In the next subsection, we outline several methods in which these gradients are used to guide mutations within the genetic algorithm. Broadly speaking, overall potential energy of the MECE system can be reduced by moving each

link in the direction of its net force. How to best utilize these gradients (or an aggregation thereof) is investigated next.

4.1.1 Directed Mutation Operators

For this investigation, we performed computational experiments on four distinct approaches, a "null" baseline method, and a "combination" approach for employing MECE gradient information within the genetic algorithm. Each experiment used the general algorithm flow outlined in Figure 2; only the nature of the Directed Mutation operator was varied. These methods used different degrees of gradient aggregation (using the net force from more than one link), and the difference between these approaches can be most simply summarized by comparing how the i^{th} updated link location $(x_i, y_i)^*$ is derived from the original location (x_i, y_i) and the net force $\vec{F}_{Net_i}$. Recall that an additional complicating factor is that the updated link position must be integer to satisfy the MECE integrality constraint.

- The **Baseline** method does not update the link locations, i.e., $(x_i, y_i)^* = (x_i, y_i)$. This method is used so that our computational timing results can fairly compare the potential speedup of gradient methods over the standard GA approach with no gradient information incorporated.
- The **Everyman** method updates each coordinate in a local manner (i.e. "every man for himself") where $(x_i, y_i)^* = (x_i, y_i) + round\left[\alpha \cdot \vec{F}_{Net_i}\right]$ and α defines a user-chosen scaling factor. This method lets every link follow its own local gradient as closely as possible while adhering to the MECE integrality constraints. Here and throughout the rest of this paper, the $round[\cdot]$ function operates so that each element of the input vector is rounded up or down independently.
- The **LP Norm** method is similar to the Everyman approach, but greater care is taken in creating a more universally applicable scaling factor. Here, $(x_i, y_i)^* = (x_i, y_i) + round\left[\beta \cdot \vec{F}_{Net_i}\right]$ such that $\beta = \frac{\alpha}{\left\|\vec{F}_{Net_i}\right\|_p}$ where α is again a user-selected scaling factor and $\|\cdot\|_p$ is the standard p-norm of the force vector where p is also chosen by the user. This idea behind this approach is that by employing the p-norm, we can find a scaling rule that works well across many problems whether the force vector is large or small.
- The **Boxcar** method uses a sliding summation of the net forces to the left and right of the i^{th} link. Here $(x_i, y_i)^* = (x_i, y_i) + round\left[\alpha \cdot \vec{G}_i\right]$ where $G_i = \sum_{j=i-b}^{i+b} \vec{F}_{Net_j}$, α is a user-selected scaling factor, and b is a user-selected "boxcar radius" that determines how far to the left and right to consider net forces for updating the i^{th} link. For example, if $b = 0$, then the method reverts to the Everyman approach. If $b = 1$, then each link is updated considering the net force on that link, as well as the force on the link to its right and left. In this manner, the net force on a *section* of the chain is employed in updating the link locations.
- The **Block Cumulative** method locates the contiguous sections (blocks) of the hanging chain with the largest cumulative force in the *x* and (independently) in the *y* direction and moves those entire sections in the desired direction. Let B_x and B_y be the blocks having the greatest absolute value cumulative *x* and *y* force, respectively. Appendix A outlines how B_x and B_y can be efficiently computed by tracking the cumulative forces

across the chain (a simple method that scales linearly with the number of links n). Once we have B_x and B_y, then the maximized cumulative forces in x and y is $CF_x = -\sum_{i \in B_x} \frac{\partial \tilde{f}}{\partial x_i}$ and $CF_y = -\sum_{i \in B_y} \frac{\partial \tilde{f}}{\partial y_i}$, respectively (i.e. the sum of the x and y forces in the B_x and B_y blocks). Finally, the position of the i^{th} link is updated as $(x_i, y_i)^* = (x_i, y_i) + m \cdot (dx_i, dy_i)$ where m is a user-entered integer move distance,

$$dx_i = \begin{cases} 0 & if\ i \notin B_x \\ 1 & if\ i \in B_x \wedge CF_x > 0 \\ -1 & if\ i \in B_x \wedge CF_x < 0 \end{cases}, \text{ and } dy_i = \begin{cases} 0 & if\ i \notin B_y \\ 1 & if\ i \in B_y \wedge CF_y > 0 \\ -1 & if\ i \in B_y \wedge CF_y < 0 \end{cases}.$$

- Finally, the **Block+Everyman** method simultaneously combines the two approaches of Block Cumulative and Everyman – in theory, coupling the possible advantages of local and aggregate gradient rules. Specifically, the link update procedure produces two new solutions – one given by $(x_i, y_i)^1 = (x_i, y_i) + m \cdot (dx_i, dy_i)$ where m, dx_i, and dy_i are defined as in the Block Cumulative method and the other given by $(x_i, y_i)^2 = (x_i, y_i) + round\left[\alpha \cdot \vec{F}_{Net_i}\right]$ where α is a scaling factor as in the Everyman method.

4.1.2 Computational Results

In this subsection, we examine the computational performances of the various Directed Mutation methods against the Baseline GA with no gradient information. All methods were applied to a MECE problem with $n = 30$ links, a spring rest length of $R = 10$, a post height of $H = 300$, and a post separation of $L = 200$. Hence the MECE must optimally locate all 30 links in a 201 by 301 grid – resulting in a search space of size 2.8×10^{143}. Figure 7 demonstrates how the various gradient methods converge towards the optimal potential energy of 6257.53.

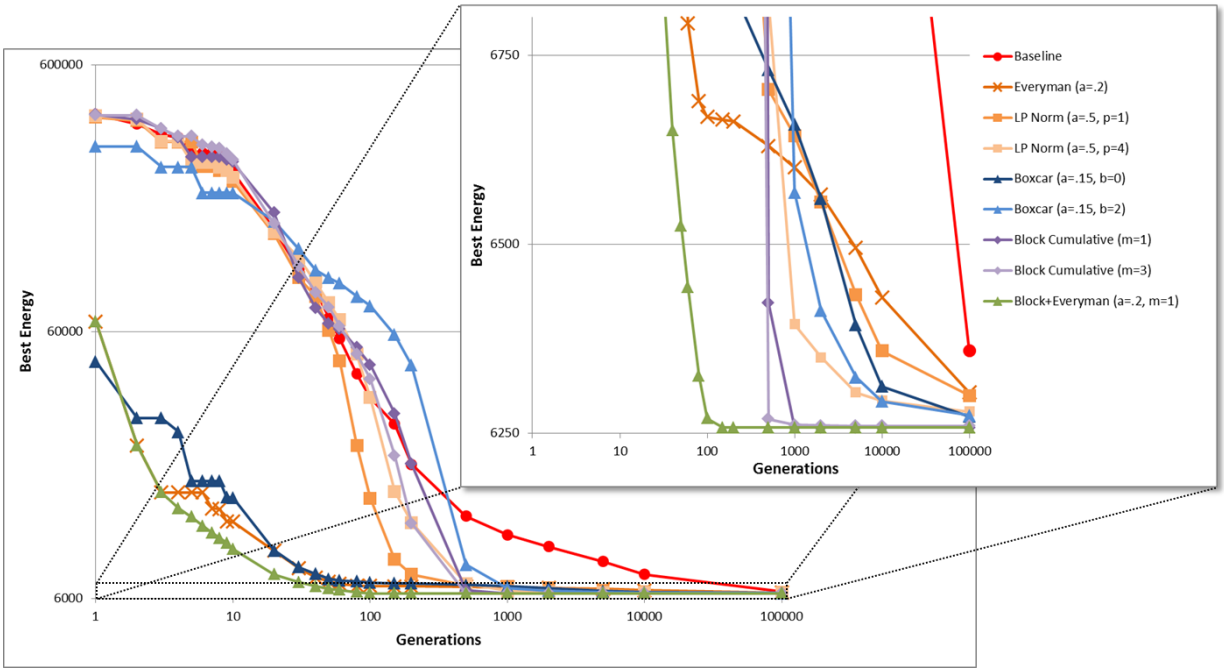


Figure 7. Computational results showing the improvement in MECE potential energy as a function of the generation (log-log scale). The inset shows convergence towards the final optimal value.

Since each of the gradient methods uses at least one user-defined parameter, we performed computation tests of a wide range of values. Figure 7 displays the one or two best realizations of those parameters for each method. Note that a few methods were able to make substantial improvement in the first generation, but the overall quality of each approach is demonstrated by the efficiency of convergence below the 6500 energy level (as shown by the Figure 7 inset). Observe that most methods perform about as well as the Baseline approach during the early generations. This is likely due to the fact that early generation solutions still have a high degree of randomness (see the "Generation 100" panel of Figure 5) that results in incoherent, looped, and kinked chain solutions. These solutions have springs that are highly stretched beyond their rest length – inducing large link forces that are unlikely to point in the direction of ultimate global optimality. Beyond generation 1000, however, every gradient approach vastly outperforms the baseline method. We attribute this delayed improvement to the eventual emergence of more coherent chain solutions (see the "Generation 1000" panel of Figure 5) whose gradients are able to more reliably guide link locations towards global optimality. Interestingly, the unaggregated methods – Everyman and LP Norm – and the slightly aggregated Boxcar method all perform roughly equivalently, while the highly aggregated Block Cumulative method converges much more quickly. By far the most successful method is the Block+Everyman approach which converges to the true optimal solution just after generation 100 – nearly an order of magnitude fewer generations than the next fastest approach.

While Figure 7 displays the convergence by generation, this does not tell the entire story. It is possible that Directed Mutation approaches spend *more time per generation* due to the increased computation needed to acquire and use the gradient information, and this slowdown per generation results in slower overall convergence when measured in CPU time. Figure 8 displays the CPU time required to reach a certain generation and indeed shows that the Baseline is faster per generation than all the Directed Mutation methods. For example, when running to a fixed 100,000 generations, the Baseline takes about 58 seconds while the Directed Mutations approaches range from 123 to 196 seconds – two to three times longer. It is not surprising that the longest time to 100,000 generations is the most complicated Block+Everyman approach. (Unfortunately, these CPU timing results were not performed on the same CPU: all Direct Mutation methods were run on a Xeon X550 @ 2.67GHz – which is no longer available for experiments – while the Baseline was run on an even more powerful Xeon E5-1650 @ 3.2GHz. Thus, we are conservative in showing the difference between the CPU timing of Baseline versus Directed Mutation in that the Baseline is faster on a per generation basis due to improved hardware.)

Nevertheless, the results in Figure 8 are very encouraging in that the CPU Time slowdown per generation is not large enough to overcome the increased efficiency in energy improvement per generation. For instance, the Baseline takes 57.7 seconds to run 100,000 generations to reach an energy level of 6,358.9. The Block+Everyman approach reaches a lower energy of 6,326.3 in only 80 generations and 0.16 seconds – giving a speedup of 1250x in generations and 360x in CPU time. Thus, it is worth spending extra CPU time per generation to get greater improvements in energy level. The next subsection devotes more careful attention to the measurement of speedup in 1, 2, and 3-objective problems.

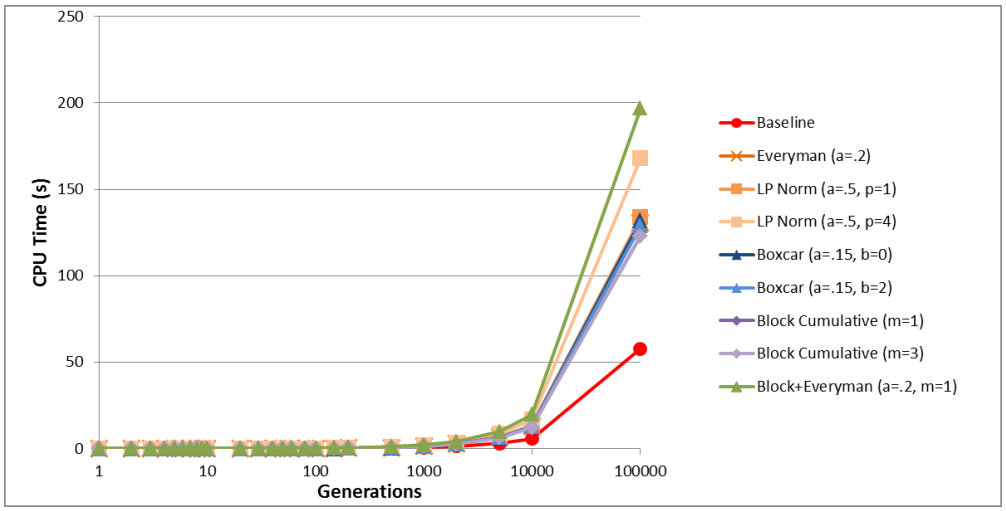


Figure 8. CPU time per generation for the Baseline and Directed Mutation methods.

In summary, for the single objective MECE problem, Directed Mutation methods provide substantial enhancement to the Genetic Algorithm. All tested methods found a lower energy level than the baseline in less CPU time and fewer generations, and the combination of *both* large block updates and individual link updates in the Block+Everyman method was the most efficient of these approaches. The only caveat, however, is that Directed Mutation gradients don't seem to provide measurable benefit during the early generations when solutions are inherently more random. But beyond these early generations, *every* gradient method we devised was a substantial improvement over the baseline.

4.2 Multi-Objective Minimum Elastic Catenary Energy Problem

The multi-objective version of the MECE problem is very similar in physical structure to the single objective problem. The system is again comprised of a chain of linked springs hanging from both ends. Now however, the orientation of the links (i.e. their locations in x and y) must simultaneously minimize two or more energy functions. There are many possible ways to form these different energy functions, and we differentiate our objective functions by using individual spring rest lengths and gravitational fields. Formally, the multi-objective MECE with m objectives is given by:

$$\begin{aligned}
\text{Multi} - \text{Objective MECE:} \quad \min \ & \tilde{f}_1(\boldsymbol{x},\boldsymbol{y}), \ldots, \tilde{f}_m(\boldsymbol{x},\boldsymbol{y}) \\
\text{s.t. } & \tilde{f}_j(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{n} p_{j,i} + \sum_{i=1}^{n-1} g_j(x_i, y_i) \\
& p_{j,i} = \left(\sqrt{(x_{i+1}-x_i)^2 + (y_{i+1}-y_i)^2} - R_j\right)^2 \\
& 0 \le x_i \le L \\
& 0 \le y_i \le H \\
& x_i, y_i \in \mathbb{Z},
\end{aligned}$$

where the nonlinear objective functions $\tilde{f}_j(\boldsymbol{x},\boldsymbol{y})$ give the total potential energy of the hanging chain under an objective-specific spring rest length R_j and gravitational potential energy function

g_j. As in the single objective MECE, the decision variables x_i and y_i give the integer Cartesian coordinates of the i^{th} link, and the constants L and H give the distance between the two supporting posts and the height of these posts, respectively.

While less physically intuitive than the single objective case, the multi-objective MECE is very important for investigating Directed Mutation methods under many competing objectives. In the previous subsection, we were able to demonstrate that Directed Mutations provide significant improvement over the baseline GA when applied to the single objective MECE. However, it is not guaranteed that these improvements will carry over into the multi-objective version. For example, the computational burden of Directed Mutations over many objectives is increased as one must now compute the gradient with respect to each objective. Also, and perhaps more importantly, each objective's gradient will generally result in a direction of improvement that conflicts with the directions suggested by other objectives. Thus, devising a link update procedure that adequately makes use of these conflicting gradient directions is a key concern.

Another difficulty arising in the multi-objective MECE is the complication in defining and measuring improvement in computational performance. In the single objective case, we could trivially compare goodness of solutions by observing which one had the lowest energy level. Thus, we could compare Directed Mutation approaches by seeing which one produced the lowest potential energy in the fewest generations or the shortest CPU time. In the multi-objective MECE, we still wish to compare Directed Mutation methods on a per-generation or per-CPU-time basis, but here each solution has *several* associated energy levels. Furthermore, unlike the single objective case where each generation has a single best energy score, here each generation consists of a *set of solutions* that best balance all the different objective energies in different ways and are all relatively non-dominated with respect to each other.

To devise a fair method that compares the relative goodness of *sets* of non-dominated solutions, consider two sets of MECE link solutions $S^1 = \{(\boldsymbol{x},\boldsymbol{y})_1^1, (\boldsymbol{x},\boldsymbol{y})_2^1, \dots, (\boldsymbol{x},\boldsymbol{y})_{k_1}^1\}$ and $S^2 = \{(\boldsymbol{x},\boldsymbol{y})_1^2, (\boldsymbol{x},\boldsymbol{y})_2^2, \dots, (\boldsymbol{x},\boldsymbol{y})_{k_2}^2\}$ and m objective functions $\tilde{f}_1$ through $\tilde{f}_m$. In general, a point $\boldsymbol{s}^1 \in S^1$ dominates a point $\boldsymbol{s}^2 \in S^2$ if $\tilde{f}_i(\boldsymbol{s}^1) \leq \tilde{f}_i(\boldsymbol{s}^2)\ \forall i = 1 \dots m$ and at least one inequality holds strictly (i.e. the solution $\boldsymbol{s}^1$ has at least as low or lower energy score in every objective function). Let D^1 equal the number of solutions in S^1 that are dominated by a solution from S^2 and D^2 equal the number of solutions from S^2 that are dominated by a solution in S^1. Using these relative domination counts, we say that S^1 is better than S^2 if and only if $D^1 < D^2$. For our computational experiments, suppose two Directed Mutations methods run for 1000 generations and produce sets S^1 and S^2, where $D^1 < D^2$. Then we would say that the first method outperformed the second over those 1000 generations.

4.2.1 Multi-Objective Directed Mutation Operators

For the multi-objective MECE, we investigated two distinct approaches for integrating multiple conflicting objective function gradients – comparing these to a Baseline method that does not use gradients. Each experiment uses the general algorithm flow in Figure 2 and implements the Block+Everyman approach (since that was the most successful single objective Directed Mutation approach); we only vary the manner in which the multiple objective gradients are employed and/or combined.

- The **Baseline** Directed Mutation operator does not update link locations. Since net force information is not used for any of the objective functions, this approach does not

calculate gradients – providing a fair computational comparison for the next two approaches (it would be unfair to calculate gradients but not use them).

- The **Independent** Directed Mutation operator takes each solution, choses one objective function uniformly at random, calculates the gradient of only the selected objective, and finally applies the Block+Everyman approach to update the link locations of that solution. The rationale for this method is that the computational savings of calculating only 1 of the *m* gradients overcomes the fact that each solution improves in only 1 objective at a time. As the optimization runs, an equal number of solutions will improve independently in each of the objective gradient directions – in theory providing a balanced improvement in all objective functions.
- The **Convex Combination** approach is the conceptual opposite of the Independent method. Here, *all* objective gradient vectors are calculated for each solution. The gradients are then aggregated together with a random, normalized, nonnegative set of multipliers (e.g., 0.5, 0.2, 0.3 in the case of 3 objectives), which produces a convex combination of the objective gradients. This convex combination of the gradients is then used to update the links using the Block+Everyman approach. The potential advantage of this method is that the convex combination of gradients might enable simultaneous improvement in multiple objectives (if the gradient directions are not in conflict), but the drawbacks are that every gradient must be computed for every solution and conflicting gradients might cancel out and leave no suggested direction.

4.2.2 Computational Results

In this subsection, we examine the computational performances of the multi-objective Directed Mutation methods against the Baseline GA with no gradient information. All methods were applied to a multi-objective MECE problem with $n = 30$ links, a post height of $H = 300$, and a post separation of $L = 200$, and up to 3 competing objective functions – each differing by choice of spring rest length and gravitational potential energy functions. Table 1 gives a summary of the 3 objective function parameters. The first objective has the longest springs and uses the standard gravitational potential energy (i.e. the gravity vector pulls downward with constant force). The second objective uses the shortest springs and has a gravitational force that pulls both down and to the right. Finally, the third objective has intermediate spring rest lengths and has a gravitational force that pulls downward but with decreasing force from left to right (each successive link has 10% less gravity than the preceding link).

Table 1. Multi-objective MECE Objective Parameters

Objective Function	Spring Rest Length	Gravitational Potential Energy Function	Minimum Energy
$\tilde{f}_1(\boldsymbol{x}, \boldsymbol{y})$	$R_1 = 20$	$g_1(x_i, y_i) = y_i$	3786.99
$\tilde{f}_2(\boldsymbol{x}, \boldsymbol{y})$	$R_2 = 5$	$g_2(x_i, y_i) = -0.5x_i + 0.866y_i$	5214.16
$\tilde{f}_3(\boldsymbol{x}, \boldsymbol{y})$	$R_3 = 12$	$g_3(x_i, y_i) = 2\left(0.9^i\right)y_i$	3700.50

Figure 9 displays the x and y coordinate link locations of the optimal hanging chain solution for each of the 3 objective functions. The optimal solution to the first objective is symmetric about the x-axis and hangs down to the "ground" due to the longer spring rest length (also

satisfying the $y_i \geq 0$ constraint). The optimal solution to the second objective hangs the highest (due to the shortest rest length) and is skewed to the right due to the gravitational force pulling both down and to the right. The optimal solution to the third objective is skewed to the left due to the decreasing gravitational force from left to right.

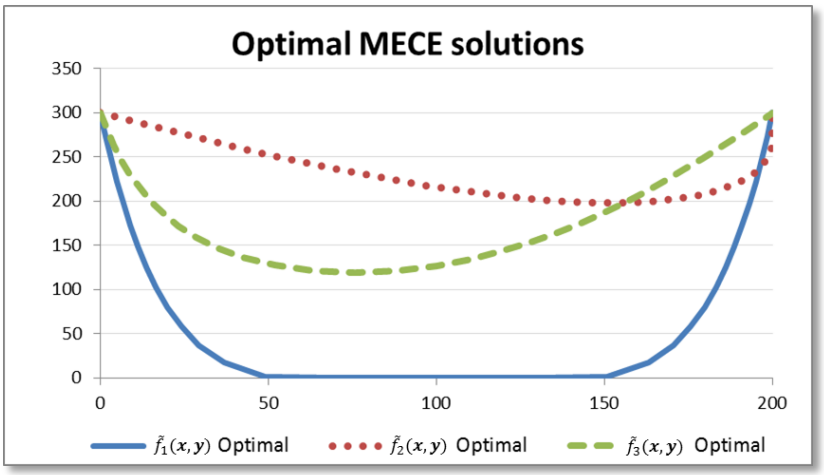


Figure 9. The optimal MECE configurations for the 3 objective functions

The goal of the multi-objective MECE is to find hanging chain configurations that simultaneously balance all three objectives. Clearly, no single solution can minimize all objective functions at once, and thus the optimization produces a set of non-dominated solutions that each provides a unique balance of the objectives. Figure 10 displays progress snapshots (at generations 300, 450, and 3000) of the GA towards the true Pareto frontier both for the Baseline (black) and the Independent Directed Mutation method (red) for the 3-objective MECE. Solutions are plotted in objective function space (i.e., every dot is a hanging chain solution plotted according to its energy level in the 3 objective functions).

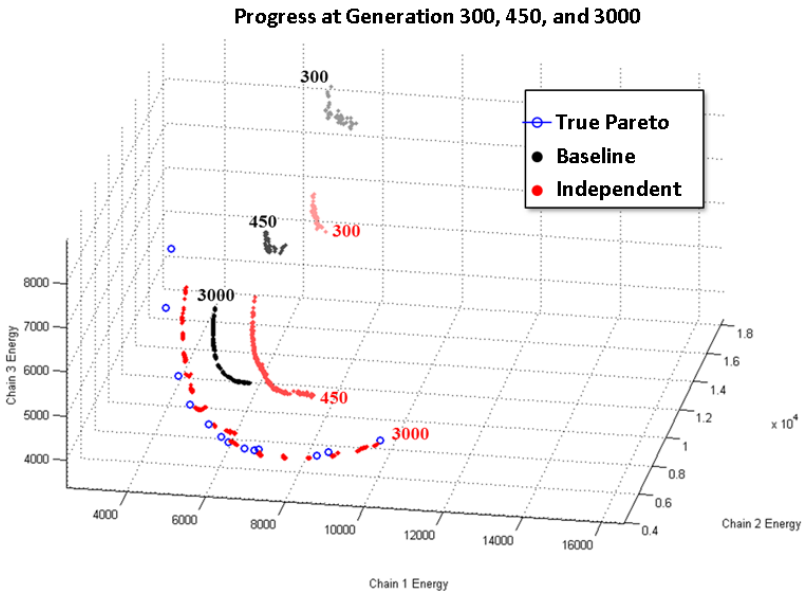


Figure 10. The progress of the Baseline and Independent Direct Mutation operators at generations 300, 450, and 3000 compared to the true Pareto frontier.

Notice that the snapshots in Figure 10 indicate a substantial speedup of the Independent method over the Baseline. At each of the generations 300, 450, and 3000, the red set of solutions (Independent approach) appears much closer to the true Pareto set than the corresponding black set (Baseline). In fact, the Independent method at generation 3000 has almost completely characterized the true Pareto set, whereas the black Baseline set is still well-removed from the Pareto and does not encompass its full extent in potential energy space. For the remainder of this subsection, we focus on more precisely measuring this speedup for MECE problems with 1, 2, and 3 objectives.

Computational experiments indicate that the Independent and Convex Combination methods perform almost identically on a per-generation basis. In other words, given that both approaches run for a set number of generations, the resulting sets of non-dominated solutions are nearly equivalent. This implies that movement in only one objective gradient at a time (i.e. the Independent approach) averages out over the generations to be equivalent to simultaneous movement in all directions (i.e. the Convex Combination method). Thus, since the Independent method is computationally cheaper per solution (only 1 gradient calculation), we employ this approach for a careful comparison of speedup over the Baseline method.

To measure this speedup, we adopt the following procedure for MECE with 1, 2, and 3 objectives. First we run the GA using the Baseline method to a set number of generations (10, 100, 1K, 10K, 100K, and 1M) – measuring the CPU time required and preserving the resulting non-dominated set of solutions (call this set S^1). Next, we run the GA with the Independent Direct Mutation method until the resulting non-dominated set (call this set S^2) surpasses the Baseline set (i.e. $D^2 < D^1$ as described in the previous subsection). We save the number of generations and CPU time for S^2 to surpass S^1 and use this information to calculate speedup ratios. For example, if the Baseline runs for 1000 generations in 20 seconds, and the Independent method surpasses this in 100 generations taking 5 seconds, then we would measure a speedup of 10x in generations (1000 generations / 100 generations) and 4x in CPU time (20 second / 5 seconds).

Figure 11 shows the speedup ratios per generation for 1, 2, and 3-objective MECE problems. The log-log plot shows a remarkably consistent speedup across the three problems, meaning that *gradient improvement retains its utility into higher dimensional problems*. The potential issue of conflicting gradient information does not seem to be apparent; gradient information is very valuable in both single objective and multi-objective MECE. More impressive is the *magnitude* of the speedup ratios. For example, when comparing the speedup over the Baseline method run to 1 million generations, the Independent Directed Mutation method shows a speedup of roughly 10,000x – meaning that the gradient method surpassed the quality of the Baseline's 1 millionth generation in only 100 generations.

The same impressive speedups are also observed in Figure 12 when comparing CPU time. Here, the 1, 2, and 3-objective speedups are slightly less consistent, as the multi-objective cases take about 10,000 Baseline generations before they start realizing speedups. But by the millionth Baseline generation, the CPU speedups of the Independent Directed Mutation are between 2000x and 12,000x.

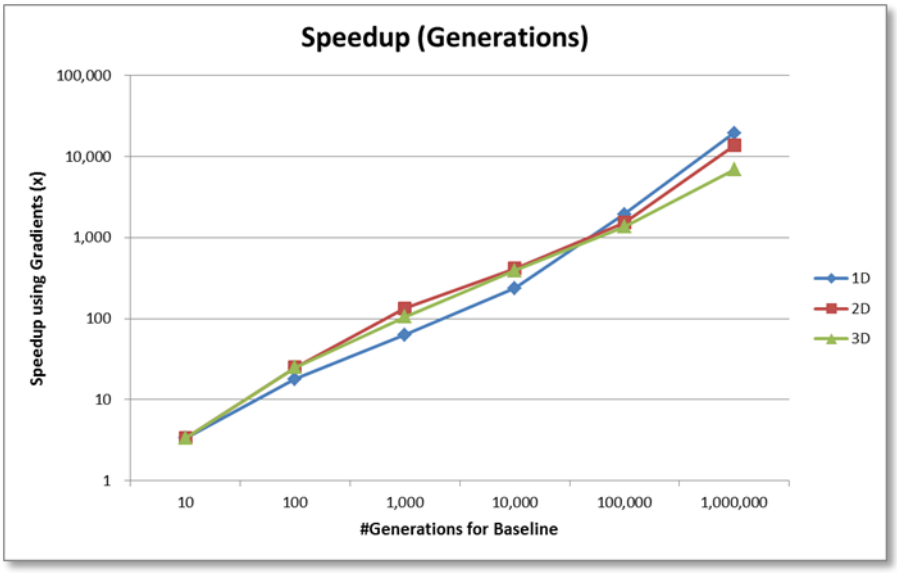


Figure 11. The speedup per generation of the Independent method vs. the Baseline

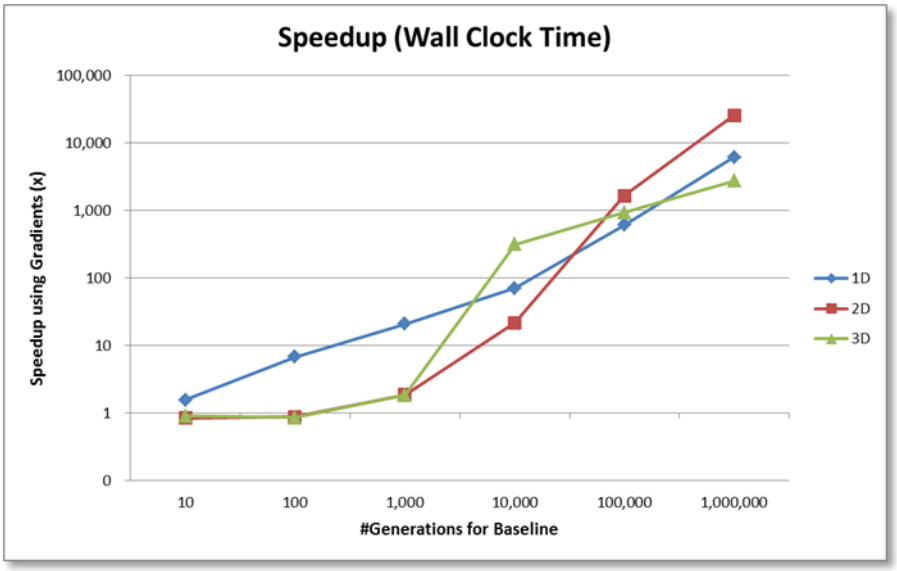


Figure 12. The speedup per CPU time of the Independent method vs. the Baseline

These impressive speedups in both generations and CPU time should likely be attributed to a combination of compounding factors. First, these speedups do indicate the great utility of gradient information for augmenting MECE solution information within the GA. Second, these speedups also exhibit the overall failure of the Baseline method to approach true optimality for the MECE problem. As described at the beginning of this section, the Baseline method must depend on fortuitous but highly unlikely combinations of mutations to coordinate multiple link updates in a desired manner. Since these lucky mutations almost never happen, the evolution of the Baseline GA tends to stall out. Because the Directed Mutations do not experience this stalling out, we observe the more and more impressive speedups as we run the Baseline to more and more generations.

5 MILITARY PORTFOLIO EXAMPLE

The hanging chain examples in the previous sections demonstrated the utility of using objective function gradient information to direct mutations in GAs when the decision variables include integer counts arrayed over time. Moreover, the latter examples showed that the benefits extend to cases where there is more than one objective function. In order to further develop the applicability of our methods to more practical situations, we sought to apply them in the case of a more complex example. In addition to being somewhat larger, the example contains system technology choices in the form of more general combinatorial variables along with dependency constraints between the technology choices.

A good example portfolio optimization problem will have more variables than the hanging chain example but not so many that it is no longer a good research vehicle. Likewise, it will be nonlinear, nonconvex, and have multiple objectives. Moreover, it will contain some non-trivial interactions between decision variables, objectives, and systems. Therefore, the following are the key features we sought to capture in our example:

- Technology options (i.e., "which type" decisions)
- Technology variables (i.e., "how many" decisions)
- Nonlinear objective functions
- Multiple, conflicting objectives
- Dependency constraints
- System-of-systems interactions

In order to convey to others the importance of this line of research, we also wanted an example that would be interesting, contemporary, relevant, and illustrative. It turns out that the military acquisition domain is ideal for developing such an example, especially when a portfolio must be considered. To begin with, military acquisition programs are large and complex. They must consider research, development, testing, and evaluation (RDT&E) costs in addition to purchase and operation and sustainment (O&S) costs. A holistic military force contains multiple interacting platforms. Its performance metrics must be evaluated in the context of one or more potential missions and must therefore be analyzed in the context of the corresponding concept of operations (CONOPS). In a sufficiently rich example, many other modeling challenges will arise. This provides opportunities to develop other modeling techniques along the way. It will also illustrate some of the thinking processes that go into creating a portfolio optimization model.

The example that we settled on is that of a fictitious military portfolio containing unmanned aerial vehicles (UAVs), vehicles, and soldiers. It builds on the idea of systems as platforms upon which new technologies are developed and deployed over time. The technologies involved are intended to be generic but reminiscent of technologies that could exist or be developed. We are, of course, abstracting away many other operational and logistical details which would be important in a real mission. Furthermore, the example may be unrealistic in certain ways, but that should not detract from the overall applicability of our research results. Exact details, realism, and technology specifics are not terribly important at this level of modeling, and we avoid many of the difficulties that adding them would entail.

5.1 Model Description and Portfolio CONOPS

The overall mission of the UAV-Vehicle-Soldier portfolio is to contain a growing insurgency in a city-sized region. The situation is modeled as follows. The region contains discrete cells of insurgents and that the number of cells is growing over time due to spread and spontaneous self-organization. Each insurgent cell is localized to within a single "sector" and the region consists of 1000 sectors. The portfolio mission lasts five years, and due to its dynamic nature, decisions on portfolio size and composition are made on a quarterly basis instead of yearly. Thus, the number of insurgent cells (I_t) and the system count decision variables are indexed by an integer t representing the quarter number in the range 1–20. Each quarter, the overall change in the number of insurgent cells depends on a fast daily dynamics which will be described next. The detailed mathematical model is developed in section 5.3.

The basic CONOPS of the portfolio is as follows. The UAVs fly daily reconnaissance missions over the region in order to gather intelligence regarding suspected locations of insurgent cells. Later that same day, squads of soldiers are dispatched to as many of the suspected locations as possible given the number of soldiers and the required number of soldiers in a squad. Each squad can head out on foot but will go in a vehicle if enough are available. The average time it takes to reach each location depends on a typical distance to all 1000 sectors. Upon arrival at their destination, each squad will determine if there is really a cell present, and if so, mitigate the threat by capturing or killing the cell members. Cells are assumed to move every day, so the probability of successfully mitigating a cell is modeled as a combination of the probability that the squad will arrive intact and the probability that the cell has not yet completed its daily mission and moved on. The third factor depends on how quickly the squads arrive relative to the average time it takes for cells to perform their daily mission.

The assumed parameters for the model as described above are summarized in Table 2.

Table 2. Global Model Parameters

Parameter	Notation	Value	Units
Initial Insurgents	I_0	50	cells
Insurgent Spread Rate	α	1/365	cells per cell per day (day^{-1})
Spontaneous Insurgents	I^{spon}	0.1	cells/day
Maximum Insurgents	$I^{\max}$	2,000	cells
Number of Sectors	N_{sec}	1,000	–
Number of Quarters	$t^{\max}$	20	–
Days per Quarter	N_Q	90	–
Insurgent Mission Time	T_I	3	hours
Typical Insurgent Distance	D	8	miles

Parameters for the platforms in the portfolio will be given in the next section.

5.2 Portfolio Decision Variables

Initially, the portfolio consists of a certain number of basic UAVs, vehicles, and soldiers. However, in order to meet the mission objectives, it is generally necessary to change the portfolio composition over time by changing the system technologies and/or varying the numbers

of each type of system. As mentioned previously, the systems in the portfolio are considered to be platforms upon which different technologies will be deployed over time. New platforms can be developed, acquired, and deployed alongside or in place of the original systems in the portfolio. Similarly, new technologies can be developed and acquired to augment or replace whatever technologies are already on each platform. For example, each original UAV has a basic sensor which could be replaced by new sensor without changing the underlying UAV platform. Changing the portfolio over time is a design optimization problem, and here we describe the decision variables. The decision variables are of two rather different types: technology options and technology variables. Treating both types of decision variables simultaneously is what we mean by holistic portfolio optimization.

5.2.1 Technology Options (Technology Development Program Choices)

We envision that there are three acquisition programs under consideration: (1) a UAV sensor upgrade program, (2) a future vehicle program, and (3) a vehicle armor program. For each contemplated technology, there is a decision variable for which of the 20 quarters the corresponding systems could first be acquired (if at all). We will call that quarter the program start date although the required corresponding program development, tooling, and first production is assumed to occur in the years leading up to that time.

For the sensor and armor program start dates, there are simultaneous one-time-only design decisions on which type to develop: sensor A, B, or C, and armor X, Y, or Z. It is assumed that there are no further design choices to be made for the future vehicle. However, we assume any newly developed armor will only work on the new vehicle, so we add dependency constraints that none of the armor programs can start before the future vehicle program.

Finally, at any time, it is possible to acquire "advanced" soldiers who have better equipment and are more highly trained than "basic" soldiers. However, this can be done without needing to introduce a corresponding program development decision variable since it is assumed that the corresponding equipment and training courses have already been developed.

The key parameters that describe these technologies are given in Table 3.

Table 3. Platform Parameters

Parameter	Notation	Units
Required sensor scan time per pass	T_P	seconds
Required number of sensor passes	N_P	–
Detections per hour per original UAV (depends on original UAV sensor type)	$u = 3600/(T_P N_P)$	hour^{-1}
Detections per hour per upgraded UAV (depends on upgraded UAV sensor type)	$u' = 3600/(T'_P N'_P)$	hour^{-1}
Number of basic soldiers to make a squad	N_S	–
Number of advanced soldiers to make a squad	N'_S	–
Average speed of an initial vehicle	v_V	MPH
Average speed of a future vehicle	v'_V	MPH
Speed reduction due to armor (depends on armor type)	v_r	MPH
Average speed of an armored future vehicle (depends on armor type)	$v''_V = v'_V - v_r$	MPH
Average speed of a basic soldier	v_S	MPH
Average speed of an advanced soldier	v'_S	MPH
Probability of arrival of unarmored vehicle (V or V')	p_0	–
Probability of arrival of armored vehicle V'' (depends on armor type)	p''_V	–

5.2.2 Technology Variables (System Counts)

Given the set of technology choices outlined above, there are up to seven distinct types of systems that could be present in the portfolio in any given quarter: (1) original UAVs, (2) upgraded UAVs, (3) initial vehicles, (4) future vehicles, (5) armored future vehicles, (6) basic soldiers, and (7) advanced soldiers. The corresponding count decision variables along with their assumed range of allowed integer values are given in Table 4.

Table 4. System Count Decision Variables

Description	Notation	Integer Range	Initial Count
count of original UAVs	U	0 – 50	10
count of *upgraded* UAVs	U'	0 – 50	0
count of initial Vehicles	V	0 – 300	20
count of *future* Vehicles	V'	0 – 300	0
count of *armored future* Vehicles	V''	0 – 300	0
count of basic Soldiers	S	0 – 1500	200
count of *advanced* Soldiers	S'	0 – 1000	0
any subset of the above count variables	$\boldsymbol{X}_t$	–	–

Each of these decision variables is implicitly indexed by t, but for clarity, we leave out the subscript. Note that the count variables for systems containing new technology should be zero before the technology program start date (and some could even be zero on or after the start date).

It should be apparent from the preceding discussion that there are many ways to represent a set of coordinated acquisition decisions for a portfolio. Our example was designed in part to illustrate a number of these techniques.

Finally, it's worth noting that the multiplicity of technology variables over time coupled with the large number of choices for each variable is the chief reason for the huge search spaces found in holistic portfolio optimization problems. Search spaces of such size severely tax a basic GA. From the data in tables above, we can compute the size of the technology variable search space: $(51\times51\times301\times301\times301\times1501\times1001)^{20} = 3.57\times10^{340}$. Including the technology options (while ignoring the dependency constraints), the overall size of the search space rises to 2.79×10^{345}.

5.3 Portfolio Metrics

A "value model" specifies the calculations for metrics by which we value the design of something. The calculations could be specified, for example, by formulas or iterative algorithms. In general, it's possible to define arbitrarily many numerical metrics as objectives in a multiobjective optimization problem. The result of the optimization is a set of Pareto optimal designs which show the optimal tradeoffs among all the metrics. It's also possible to group metrics together in various ways to simplify the trade space. For our military portfolio example, we wanted to keep it relatively simple for trade space visualization proposes while still demonstrating a general capability. Thus we chose to group all our metrics according to costs and performance. The costs group contains all the metrics related to cost while the performance group contains all the rest.

5.3.1 Costs Group

The costs group contains the cost in each of the 20 quarters as well as the total cost over the entire program. To simplify the accounting, inflation and discount factors are assumed to cancel out. Quarterly costs will be called "budgets" to reflect the fact that they usually have hard limits. The total cost over the entire timeframe is simply the sum of the 20 quarterly budgets. It would be a redundant metric except for the fact that it is treated separately by the GA's fitness assessor as described below.

The costs group also contains a "phantom cost" metric which is included as a technicality to improve the convergence of the GA. Before a given program start date, the corresponding system counts are "phantoms" that may not be zero although they are always interpreted as zero in the metric calculations. This means that there are many equivalent solutions, and the search space is much larger than it really needs to be. The phantom cost metric is simply the additional cost that would be incurred if the phantom systems were real (without changing the RDT&E costs since they only apply to the real systems). Its intended effect is to force any phantom system counts to zero and thereby reduce the size of the search space and speed convergence.

5.3.2 Performance Group

Recall that the overall mission of the portfolio is to contain an insurgency, so we would like to define a single "insurgency" metric which captures the performance of the portfolio over the entire timeframe. First, we need a model of what is bad about an insurgency. To that end, each day, each cell is considered to bring a fixed amount of harm to the region. So the insurgency metric should take into account the number of insurgent cells as well as the amount of time they are present. When they are present is not considered important. With these considerations, a reasonable quantity for the insurgency is simply the sum of the number of cells over the 20 quarters. Additional metrics could be developed to represent goals of other missions of the portfolio.

In addition to the insurgency metric, we include the technology dependency constraints in the performance group. Solutions which violate these constraints have lower fitness, so the GA will favor solutions that satisfy the most dependency constraints.

5.3.3 Fitness and Metric Formulations

Candidate designs in a GA evolve depending on fitness functions rather than directly on the metrics themselves. Each metric group is scored by a fitness which is a weighted sum of the fitness associated with each component metric. The weights are proportional to the length of the time with which they are associated; specifically, either one quarter or 20 quarters. The individual fitness functions for each metric are parameterized by a "limit" value and an "objective" value.[1] The limit is chosen to represent a value of the metric which is considered to be "poor." The objective is chosen to represent a value of the metric which is considered to be "good." Dependency constraints are either satisfied or not so do not require any parameter adjustments.

[1] Strictly speaking, there are more than two parameters for each fitness function, but the other parameters are left with their default values.

In order to use gradient-directed mutations, we demand that every metric be formulated piecewise in terms of closed-form formulas. This may not be possible since metrics are sometimes defined and calculated in terms of highly iterative numerical routines of some sort (Monte Carlo integration, agent-based simulations, or numerical integration of differential equations). However, even in some of these cases it may be possible to calculate gradients as part of the simulation (for example, by using automatic differentiation).

Fortunately, if we base our metric formulations as much as possible on economics and physics, we can frequently develop closed-form, analytic approximations for the metrics. In that case, it is possible to formally differentiate all the metrics with respect to all the system counts. Since the fitnesses are smooth functions of the metrics, we can then in turn find the gradient of the fitness of each group by using the chain rule. Where the derivatives are discontinuous, we can pick an arbitrary element of the subgradient (such as a sequential average of the gradients from each piece).

5.3.4 Cost and Budget Formulation

The total cost over the entire timeframe is simply the sum of the 20 quarterly budgets. The budget in each quarter is modeled in turn as the sum of three types of cost contributions: (1) initial system purchases, (2) quarterly costs of systems in inventory, and (3) RDT&E costs. Initial system costs include the platform and any corresponding mounted technologies. The quarterly costs include O&S costs as well as system replacement costs based on the initial cost amortized over the average lifetime of the system. RDT&E costs are only incurred if the program is started sometime during the 20 quarters, and if so, they are modeled as the same total being evenly spread over the quarters up to and including the program start date. Hence, the quarterly RDT&E costs can be reduced by extending the development time.

If assets are upgraded in a given quarter, only the newly mounted technologies contribute to the budget since the underlying platform has already been paid for. If the portfolio is divested of systems or technologies, their purchase costs are simply lost. In other words, we assume neither proceeds from sale nor disposal costs. This means it probably does not make financial sense to reduce the number of systems in one quarter and buy them soon thereafter: it is probably cheaper to pay the intervening quarterly costs.

Since the marginal cost of buying one more system is zero or some non-zero constant depending on whether the asset counts are falling or rising, it means that the budgets must be expressed as piecewise multilinear functions of the number of assets. However, both the budgets and total costs can still be formally differentiated with respect to system counts on each multilinear piece. At the boundaries where the pieces come together, we use an average of the derivatives from each piece.

The fitness of the costs group is the sum of the fitnesses of the 20 budgets and the fitness of the total cost. The budget limits are modeled as being hard, but the fitness of each one is weighted approximately $1/20^{th}$ as much as the fitness of the total cost (only approximately because on an actual calendar, the quarters may vary in length by a day or two). The cost limit is soft and is set to only 80% of the sum of the budget limits. This makes the GA tend to prefer portfolio designs in which not all budgets are pegged at their absolute limits.

5.3.5 Performance Formulation and System of Systems Physics

While the budget and cost calculations are straightforward to model based on economics (especially since we are not doing any discounting), the same is not true for performance metrics. To the extent possible, we would like performance metrics to be based on an analysis of the "physics" of the portfolio: that is, they should reflect quantitative cause and effect relationships. It helps to make sure such relationships are naturally formulated in a way that is dimensionally consistent between factors and terms. Ideally, dimensional consistency can be attained using parameters that can, at least in principle, be measured rather than parameters that must be tuned. If we are successful, we will obtain performance metrics that are formulated piecewise in terms of closed-form formulas. Each piece will almost always be an analytic formula expressed in terms of the portfolio decision variables, and it will then be possible to use gradient-directed mutations.

Building on the CONOPS described above, we will now formulate the insurgency metric based on an analysis following the guidelines above. We are given the initial number of insurgent cells, I_0, and we need to derive an expression for the number of cells in each subsequent quarter as a function of the portfolio composition over time. We think of I_t as being the number of cells at the end of quarter t. It will depend on the number of cells at the beginning of the quarter (i.e., from the end of the previous quarter) along with the daily dynamics implied by the composition of the portfolio during the quarter: $I_t = I_t(I_{t-1}, \boldsymbol{X}_t)$.

We start by assuming that when the squads are dispatched to mitigate cells, the probability of a successful engagement has a decaying exponential dependence on the time taken for the UAVs to detect the cells plus the time to travel to the cells.

$$P(I_{t-1}, \boldsymbol{X}_t) = P(I_{t-1}, U, U', V, V', V'', S, S') = \exp\left(-\frac{T_D + T_T}{2T_I}\right).$$

The arguments to the function reflect the fact that both the detection time and travel time depend in detail on the composition of the portfolio. The total time to detect all cells only depends on the number and type of UAVs along with the number of sectors. It is simply the number of sectors divided by the total rate of cell detection by all UAVs:

$$T_D(\boldsymbol{X}_t) = T_D(U, U') = \frac{N_S}{uU + u'U'}.$$

In contrast, the travel time does not depend on the UAVs. It depends on the number of insurgent cells as well as the number of vehicles, and soldiers: $T_T(I_{t-1}, \boldsymbol{X}_t) = T_T(I_{t-1}, V, V', V'', S, S')$. We break it down in the following paragraphs.

The total distance traveled by all squads is the number of squads times the given average distance traveled by each squad: $Q \times d$. It can be approximated by the average travel time multiplied by a sum total average velocity of all squads. This is a reasonable approximation if the faster squads tend to be dispatched to the more distant cells so that the travel times are all about the same. Rearranging gives the average travel time:

$$T_T(\boldsymbol{X}_t) = \frac{Qd}{v_T(\boldsymbol{X}_t)}.$$

The sum total average velocity $v_T(V, V', V'', S, S')$ of all the squads depends on the number of vehicles and soldiers and can be derived as follows. At most one squad is sent out to each cell, so the number mobilized squads is given by

$$Q(I_{t-1}, \boldsymbol{X}_t) = \min\left(I_{t-1}, \frac{S}{N_S} + \frac{S'}{N_S'} \right). \tag{1}$$

Squads are always put in vehicles when possible, so given the total number of vehicles, $V_T(\boldsymbol{X}_t) = V + V' + V''$, the number of squads in vehicles will be

$$Q_V(I_{t-1}, \boldsymbol{X}_t) = \min\big(Q(I_{t-1}, \boldsymbol{X}_t), V_T(I_{t-1}, \boldsymbol{X}_t)\big) = \min\left(I_{t-1}, V_T, \frac{S}{N_S} + \frac{S'}{N_S'} \right).$$

The remainder of the squads will be on foot:

$$Q_F(I_{t-1}, \boldsymbol{X}_t) = Q(I_{t-1}, \boldsymbol{X}_t) - Q_V(I_{t-1}, \boldsymbol{X}_t).$$

The total speed of all squads will be determined by the average of the speeds of the vehicle and foot squads weighted according to the fraction of each. Therefore, we need the following fractions where $S_T(\boldsymbol{X}) = S + S'$ is the total number of soldiers:

$$\begin{aligned} f_V(\boldsymbol{X}) &= V/V_T \\ f_V'(\boldsymbol{X}) &= V'/V_T \\ f_V''(\boldsymbol{X}) &= V''/V_T \\ f_S(\boldsymbol{X}) &= S/S_T \\ f_S'(\boldsymbol{X}) &= S'/S_T. \end{aligned}$$

Furthermore, we model the effect of the reduced probabilities of arrival (p_0 and $p_V{}''$) as a corresponding reduction in average speed. Putting this all together gives the total average speed of all squads:

$$v_T(\boldsymbol{X}_t) = [(v_V f_V + v_V' f_V')p_0 + v_V'' f_V'' p_V'']Q_V + [v_S f_S + v_S' f_S']Q_F.$$

The upshot of the forgoing discussion is that we have analytic expression for $T_D(\boldsymbol{X}_t)$ and $T_T(\boldsymbol{X}_t)$ and therefore have an analytic expression for $P(\boldsymbol{X}_t)$.

By using the parameters in Table 2 we can derive an equation that gives the daily change in the number of cells. First consider the increment in the insurgency in the first day of quarter t (i.e., starting with I_{t-1} insurgents from the previous day). Assuming the daily mitigation happens before the daily spread and spontaneous generation, we can write

$$\begin{aligned} \Delta I_d(I_{t-1}, \boldsymbol{X}_t) &= I^{\text{spread}} + I^{\text{spon}} \\ &= \alpha(I_{t-1} - PQ) + I^{\text{spon}} \\ &= \alpha\big(I_{t-1} - P(I_{t-1}, \boldsymbol{X}_t)Q(I_{t-1}, \boldsymbol{X}_t)\big) + I^{\text{spon}} \end{aligned}$$

If we assume the subsequent change in the number of cells is approximately the same for all N_Q days in the quarter, we obtain

$$\begin{aligned} I_t(I_{t-1}, \boldsymbol{X}_t) &= (I_{t-1} - PQ) + N_Q[\alpha(I_{t-1} - PQ) + I^{\text{spon}}] \\ &= \big(1 + N_Q\alpha\big)\big(I_{t-1} - P(I_{t-1}, \boldsymbol{X}_t)Q(I_{t-1}, \boldsymbol{X}_t)\big) + N_Q I^{\text{spon}}. \end{aligned}$$

In any given military theatre, the number of cells would not grow without bound because of the limited regional population along with other unspecified feedbacks. To reflect these considerations, we add a cap of $I^{\max}$ to the dynamics by fiat:

$$I_t(I_{t-1}, \boldsymbol{X}_t) = \min\big(I^{\max}, \big(1 + N_Q\alpha\big)\big(I_{t-1} - P(I_{t-1}, \boldsymbol{X}_t)Q(I_{t-1}, \boldsymbol{X}_t)\big) + N_Q I^{\text{spon}}\big). \qquad (2)$$

Finally, this equation is used recursively to calculate the insurgency metric which is the total number of cells over all quarters: $I = \sum_t I_t$.

Unfortunately, the formulation for ΔI_d above only considered the daily change in the number of cells *after* mitigation has taken place which means ΔI_d is always ≥ 0. The resulting dynamics given by equation (2) corresponds to a CONOPS in which the squads are only dispatched in the first day of each quarter, and then the growth in the number of cells is the same every day throughout the quarter.

The overall change in the number of cells in the first day should have been calculated as follows:

$$\begin{aligned} I_d(I_{t-1}, \boldsymbol{X}_t) &= I^{\text{unmitigated}} + I^{\text{spread}} + I^{\text{spon}} \\ &= (I_{t-1} - PQ) + \alpha(I_{t-1} - PQ) + I^{\text{spon}}. \end{aligned}$$

Thus,

$$\Delta I_d(I_{t-1}, X_t) = I_d - I_{t-1} = -PQ + \alpha(I_{t-1} - PQ) + I^{\text{spon}}.$$

The presence of the extra $-PQ$ term means that ΔI_d can be < 0. This makes more sense and reflects how effective the daily CONOPS could be.

Note from equation (1) that the number of squads Q is either proportional to I_{t-1} or capped by the constant number of soldiers in quarter t, so the daily dynamics formula is bounded by terms linear in I_{t-1} without any negative feedback terms. Therefore, if it is iterated recursively, the number of cells will not change linearly throughout the quarter but will grow or shrink exponentially. After 90 iterations, essentially any exponential divergence will swamp reality. This exposes the fact that we have not captured a realistic CONOPS for this portfolio. In spite of these shortcomings in the derivations above (or perhaps because of them), equation (2) actually gives reasonable-looking behavior. So for purposes of illustration in this research, it will suffice as a *definition* of the insurgency metric in what follows.

5.3.6 Model Parameters

The sections above laid out the notation for model parameters and decision variables along with formulas for the performance metrics. Calculation of cost metrics was described in section 5.3.4. To demonstrate an optimization using the example, we need data for parameters and initial system counts. Actual data could be used if this were a real example, but since our example was only designed for purposes of research and illustration, we generated notional data.

Cost data are separate for each platform or technology deployed on a platform. Each asset has a unit purchase price and yearly O&S costs. Technologies that have yet to be developed also have RDT&E costs which are spread out evenly during the years leading up to the first acquisition. Finally, each asset is assumed to have an average time over which it needs to be replaced whether it be due to age, damage, or retirement. The yearly replacement cost is estimated as the initial price divided by the mean loss time. The corresponding data are given in Table 5 and Table 6.

Table 5. UAV and Soldier Cost Data

Cost Item	UAV	Sensor 0	Sensor A	Sensor B	Sensor C	Basic Soldier	Advanced Soldier
Price (@, $)	3,000k	100k	200k	500k	1,000k	20k	40k
O&S Cost(@, $/qtr)	1,000k	5k	10k	30k	50k	50k	70k
Mean Loss Time (qtr)	8	16	12	10	10	15	12
RDT&E Cost	0	0	4,000k	7,000k	10,000k	0	0

Table 6. Vehicle and Armor Cost Data

Cost Item	Initial Vehicle	Future Vehicle	Armor 0	Armor X	Armor Y	Armor Z
Price (@, $)	1,000k	1,500k	0	30k	60k	100k
O&S Cost(@, $/qtr)	30k	20k	0	0	0	10k
Mean Loss Time (qtr)	20	25	∞	∞	∞	5
RDT&E Cost	0	20,000k	0	1,000k	2,000k	4,000k

No data is required to describe the UAVs themselves, but for the various UAV sensors, we need to know scan times and the number of passes required. For soldiers, the number required to form a squad as well as the squad foot speed are parameters. These data are given in Table 7.

Table 7. UAV Sensor and Soldier Technology Data

Technology Specification	Sensor 0	Sensor A	Sensor B	Sensor C	Basic Soldier	Advanced Soldier
Scan Time (s)	20	18	18	20	–	–
# Passes Required	5	4	3	2	–	–
Average Speed (MPH)	–	–	–	–	3	5
Soldiers per Squad					6	4

For vehicles and any associated armor (Armor 0 means none), we need to know average speed and arrival probability data. These are given in Table 8.

Table 8. Vehicle and Armor Technology Data

Technology Specification	Initial Vehicle	Future Vehicle	Armor 0	Armor X	Armor Y	Armor Z
Average Speed (MPH)	25	33	–	–	–	–
Speed Reduction (MPH)	–	–	0	4	8	6
Arrival Probability	–	–	0.5	0.7	0.95	0.99

5.4 Implementation Details and Results

The model described above was implemented using Sandia's Technology Management Optimization (TMO) application [15, 16] which is based on the same JEGA software used for the hanging chain experiments. It adds a graphical user interface (GUI) with a number of convenient features which facilitate setting up optimization problems for solution using a GA. TMO is especially suited to optimization problems with a time-based component such as our holistic portfolio optimization problems, and one of its main outputs shows solutions in a form that looks much like a Gantt chart.

A key feature of TMO is the ability to use an "external evaluator" which is custom software to evaluate complex metrics of each new offspring in the evolving population of trial solutions. A recent addition to the external evaluator capability resulting from our research is the ability to generate arbitrary new offspring in addition to evaluating them. A basic GA generates new population members using the processes of random initial generation, random mutation, and random crossover. The upgraded external evaluator capability allows us to generate new population members using any algorithm we choose—random or deterministic—including algorithms that use directed mutations, with or without taking advantage of gradient information.

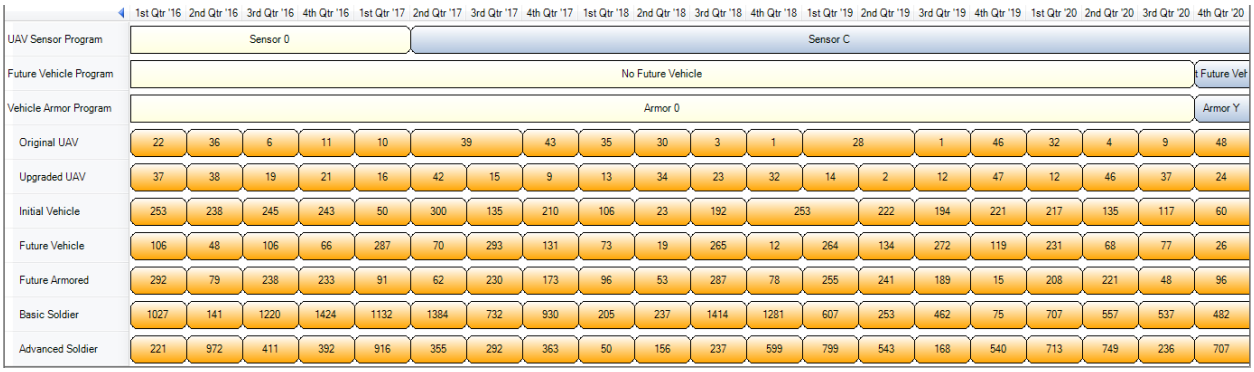
5.4.1 Basic Military Portfolio Implementation

For a baseline experiment, the TMO GUI was used to set up a 20 quarter timeframe while decision variables and dependency constraints were defined corresponding to those described in section 5.2. An external evaluator was written to read in the data in Tables 5–8 from a spreadsheet and then evaluate the budget, cost, phantom cost, and insurgency metrics as formulated in section 5.3. The dependency constraint objectives are evaluated internally by TMO. Finally, budgets, total cost, and phantom cost were assigned to the costs group while insurgency and dependency constraints were assigned to the performance group. Limit and objective parameters for each metric were entered into according to Table 9.

Table 9. TMO Parameters

TMO Parameter	Value	Units
Budget Limit	50,000k	$/quarter
Budget Objective	10,000k	$/quarter
Cost vs. Budget Efficiency	80%	–
Total Mission Cost Limit	800,000k	$
Total Mission Cost Objective	160,000k	$
Insurgency Limit	2,000	cell-quarters
Insurgency Objective	200	cell-quarters

An initial population of candidate portfolio decisions is randomly generated by TMO. An example of one member of this initial population is shown in Figure 13. Time is broken out horizontally into 20 consecutive quarters. The first three rows show the acquisition program decisions while the remaining seven rows show the integer system count decisions.



	1st Qtr '16	2nd Qtr '16	3rd Qtr '16	4th Qtr '16	1st Qtr '17	2nd Qtr '17	3rd Qtr '17	4th Qtr '17	1st Qtr '18	2nd Qtr '18	3rd Qtr '18	4th Qtr '18	1st Qtr '19	2nd Qtr '19	3rd Qtr '19	4th Qtr '19	1st Qtr '20	2nd Qtr '20	3rd Qtr '20	4th Qtr '20
UAV Sensor Program	Sensor 0					Sensor C														
Future Vehicle Program	No Future Vehicle																			Future Veh
Vehicle Armor Program	Armor 0																			Armor Y
Original UAV	22	36	6	11	10	39	43	35	30	3	1	28	1	46	32	4	9	48		
Upgraded UAV	37	38	19	21	16	42	15	9	13	34	23	32	14	2	12	47	12	46	37	24
Initial Vehicle	253	238	245	243	50	300	135	210	106	23	192	253	222	194	221	217	135	117	60	
Future Vehicle	106	48	106	66	287	70	293	131	73	19	265	12	264	134	272	119	231	68	77	26
Future Armored	292	79	238	233	91	62	230	173	96	53	287	78	255	241	189	15	208	221	48	96
Basic Soldier	1027	141	1220	1424	1132	1384	732	930	205	237	1414	1281	607	253	462	75	707	557	537	482
Advanced Soldier	221	972	411	392	916	355	292	363	50	156	237	599	799	543	168	540	713	749	236	707

Figure 13. A random initial portfolio.

This figure represents a particular set of decisions, not necessary good ones (we sometimes refer to such a set of decisions as a "portfolio" or a "solution" even though it's really only a possible plan for a future portfolio). In this instance, Sensor C is chosen for development and can be added to the UAVs starting in the second quarter of 2017. Similarly, the future vehicle and Armor Y for it can be purchased in the last quarter of 2020. The system counts are randomly chosen from the ranges defined in Table 4. Recall that any system counts before the corresponding program start quarter are phantoms and therefore ignored for budget, cost, and performance purposes. Clearly this portfolio is far from optimal because of the repeated investment and divestment decisions for all seven systems.

5.4.2 Basic Genetic Algorithm Results

To establish a baseline for performance, we used the basic GA built into TMO to perform the multiobjective optimization of this holistic portfolio. Figure 14 shows the trade space of fitness values of the solutions in the current best estimate of the Pareto efficient frontier. This result took about 12 hours to evolve on a current desktop computer. Each dot represents a non-dominated solution meaning that the cost fitness cannot be improved without reducing the performance fitness and vice versa. The solutions represented by green dots are "feasible" meaning that all dependency constraints are satisfied and all of their metrics are at or better than the limit values defined in Table 9. The solutions represented by the red dots have at least one constraint or limit violation. The large green dot represents the single best solution as determined by the highest sum of the costs fitness and performance fitness.

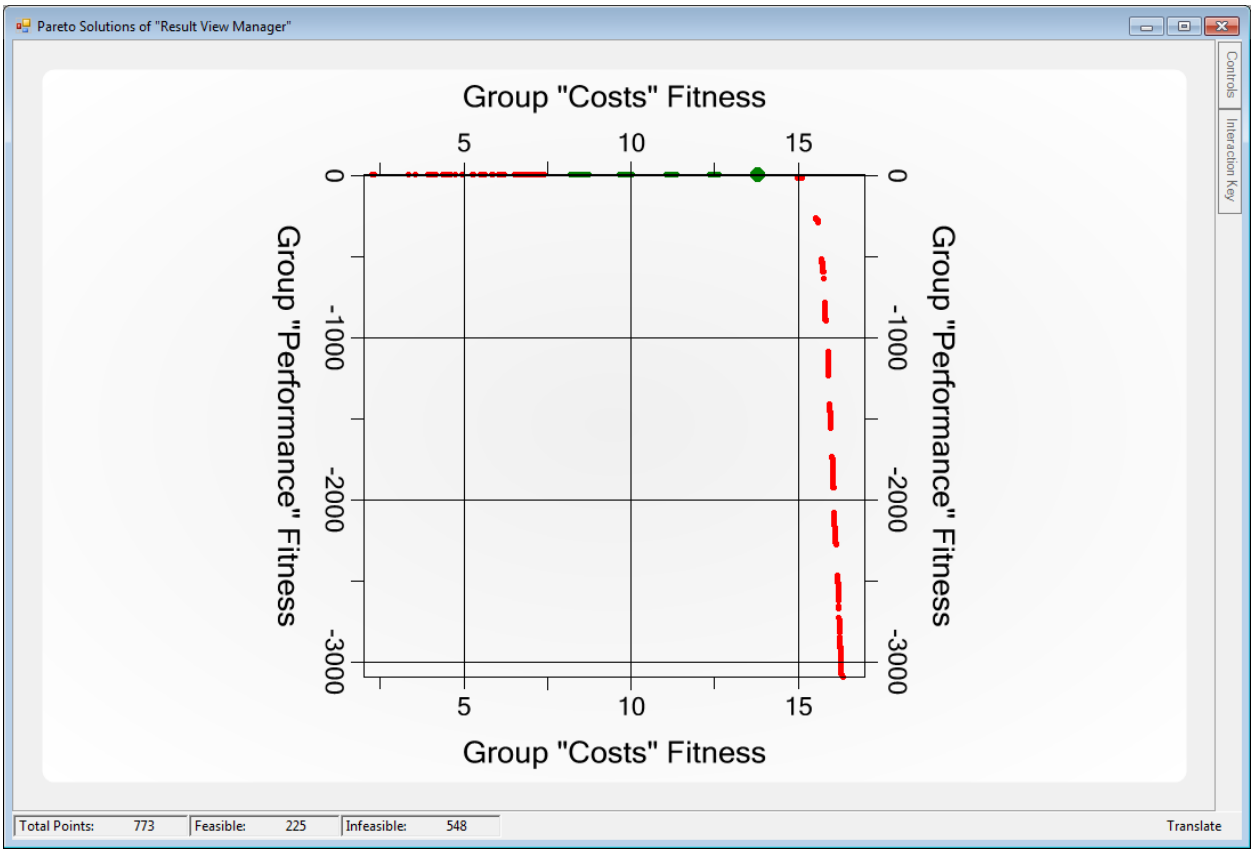


Figure 14. Optimized trade space of non-dominated holistic portfolio designs.

The large green dot in Figure 14 corresponds to the decisions shown in Figure 15. It's worth pointing out a few salient features of this set of decisions. Sensor C will be developed and deployed to upgrade all UAVs in the first quarter with no original UAVs left in the portfolio at any time. Since neither of the other two acquisition program decisions was exercised, the non-zero counts of the Future Vehicle and Future Armored Vehicle are phantoms. Those numbers are small because the phantom cost metric tends to force phantom counts to zero (all phantom counts should be identically zero in a truly Pareto optimal solution). With a few exceptions, system count profiles are lower, monotonically decreasing, and much smoother than they were initially. These features arise from the tendency to reduce budgets and overall costs while still maintaining an acceptable level of performance.

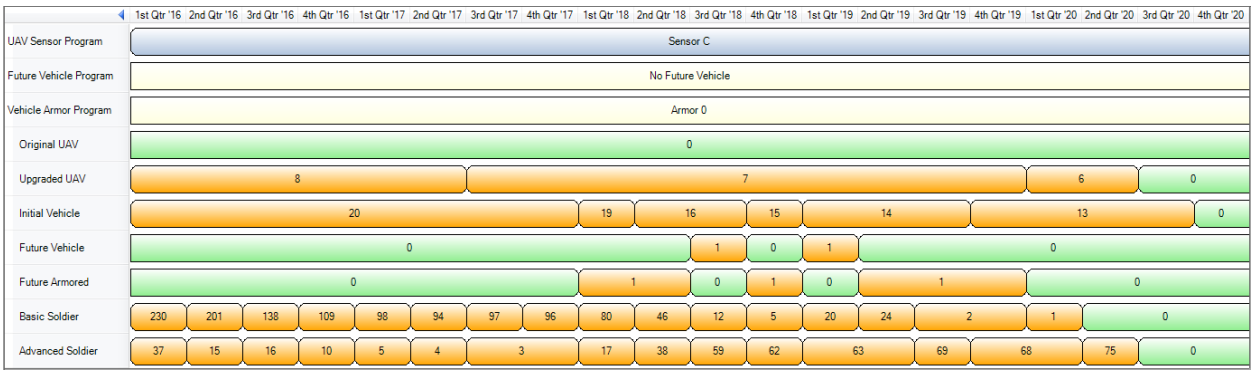


Figure 15. A partially optimized portfolio.

Figure 16 shows the same portfolio over time with the quarterly budgets lined up below (in blue). The total cost is the sum of the budgets (i.e., the area under the curve). The horizontal red line in the budget graph is the budget limit for every quarter while the horizontal green line is the budget objective for every quarter. Since all the budgets are below the red line, they are considered feasible. The expenditure in the first quarter is much larger than the rest because it includes the full RDT&E cost of Sensor C ($10,000k from Table 5). The budget is held at an intermediate level during most of the timeframe. The budget falls to zero by the end because of the decisions to divest all assets.

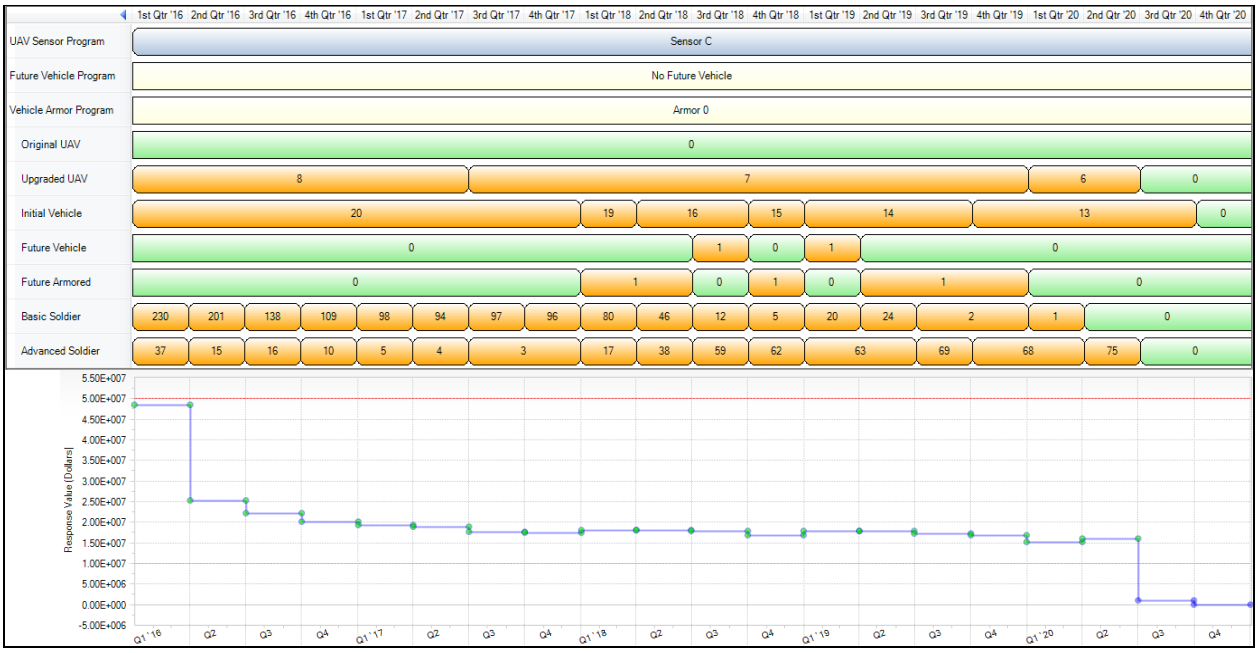


Figure 16. Quarterly budgets aligned to the portfolio.

In the process of calculating the integrated insurgency metric, the external evaluator also records the number of cells in each quarter. The results are shown in Figure 17. It shows the same portfolio over time as before with the quarterly number of insurgent cells aligned below (in green). Recall that the insurgency metric is the sum of the number of cells (i.e., the area under the curve). There are no limits or objectives for the quarterly numbers of cells, just for the total insurgency size. In this scenario, the number of cells is quickly driven down to fewer than 20. Note that the insurgency limit of 2000 in Table 9 is equivalent to an average of 100 cells in each of the 20 quarters while the insurgency objective of 200 is equivalent to an average of 10 cells in each quarter. As the end of the timeframe approaches, the area under the curve is small enough that the optimization gives up on maintaining the portfolio and divests all its assets which saves on costs. The portfolio over time is still optimal in the sense that total insurgency size is held within desired levels while keeping a favorable balance with costs. This behavior seems surprising at first, but it also makes sense in light of the way the multiobjective optimization was formulated. It also recapitulates the historical observation that pulling out of a peacekeeping mission may allow undesirable elements to resurface.

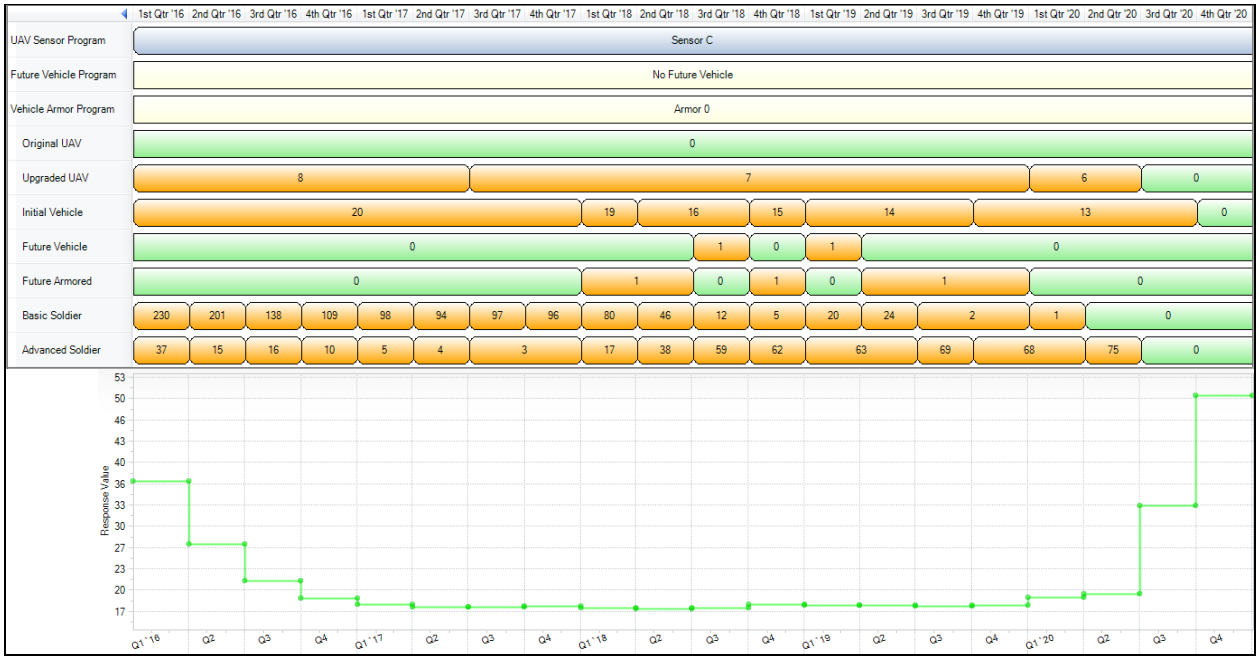


Figure 17. Number of insurgent cells aligned to the portfolio.

5.4.3 Gradient Calculations and Interpretations

The single best solution above took 12 hours to compute and still contains decisions that are clearly suboptimal. We know they are suboptimal because we can intuitively see that changing the system counts in certain ways will improve one or more of the metrics without hurting any others. In particular, we know we would do well if we change the system counts in the general direction determined by the partial derivative of the phantom cost with respect to those system counts. There are subtleties to be sure, but by using the gradient information judiciously, directed mutations can be designed to target poor decisions in the system counts.

Of course, in order to generate gradient-directed mutations, we need to calculate the partial derivatives of all the metrics with respect to all of the system counts. The calculations are too extensive to spell out in the main line of the text, but an example is given in Appendix B. There we consider the contribution of the 20 original UAV counts (U_t) to 21 metrics: the 20 quarterly budgets and the total cost. The partial derivatives of those 21 metrics with respect to U_t are calculated at the same time as the metrics.

To get an idea of what the resulting partial derivatives look like and how they work, an example is shown in Figure 18. The top part of the figure shows U_t over the 20 quarters. The lower part of the figure is aligned with the top and shows the instantaneous change in the total cost of the portfolio per unit addition of UAVs in any given quarter ($\partial C/\partial U_t$). Note how the change in portfolio cost depends on which quarter the UAV is added. Perhaps surprisingly, the change in cost is not even always positive.

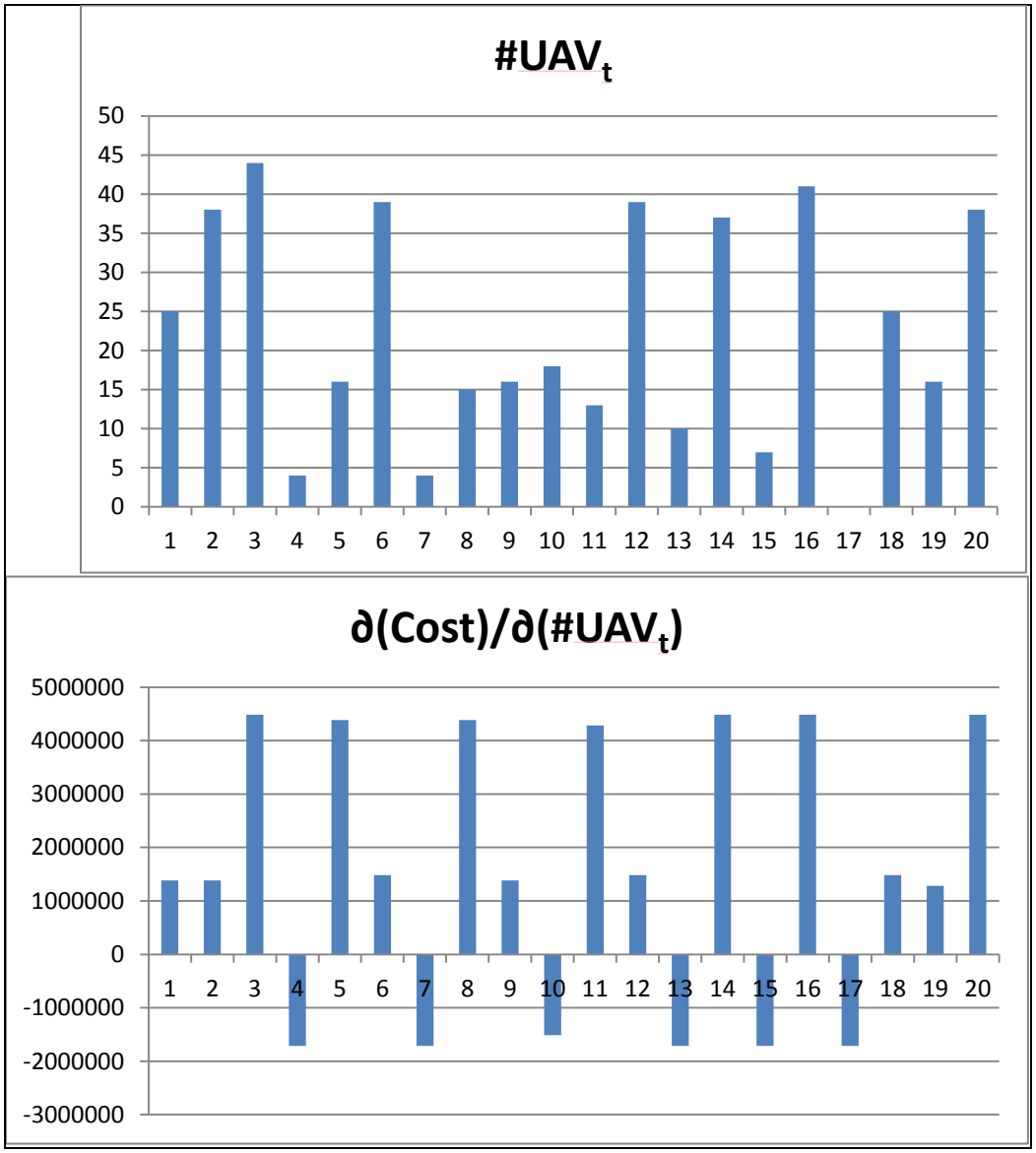


Figure 18. Partial cost gradient w.r.t. # UAVs aligned to the # UAVs.

The typical values of the individual partial derivatives can be explained as follows. Refer back to the costs in the UAV and Sensor 0 columns of Table 5. If the number of UAVs in any given quarter is a peak, higher than either surrounding quarter, the full cost of the UAV and its sensor are incurred: ($\$3000\text{k} + \$1000\text{k} + \$3000\text{k}/8 + \$100\text{k} + \$5\text{k} + \$100\text{k}/16 = \$4486.25\text{k}$). If the number of UAVs in any given quarter is a valley, lower than either surrounding quarter, then the purchase cost of the UAV is saved while the yearly costs are still incurred ($-\$3000\text{k} + \$1000\text{k} + \$3000\text{k}/8 - \$100\text{k} + \$5\text{k} + \$100\text{k}/16 = -\$1713.75\text{k}$). If the number of UAVs in any given quarter is a slope, rising or falling, there are no purchase savings and the extra yearly costs are incurred ($\$1000\text{k} + \$3000\text{k}/8 + \$5\text{k} + \$100\text{k}/16 = \$1386.25\text{k}$). Deviation from these typical values happen because, for example, the total number of UAVs (original plus upgraded) might be a peak yet the number of original UAVs is a slope meaning that the cost of the original sensor does not contribute to the total cost gradient in that quarter. Another way it can happen is if the cost function is at the boundary between two pieces of the domain in which case an element of the subgradient is chosen (generally the average of the gradients of the two pieces).

The partial derivatives are the components of the total cost gradient, and they tell us in general what to do to cut cost. Directed mutations should tend to move the UAV numbers by an amount proportional to the direction and magnitude of the force (which is opposite to the cost gradient). In Figure 18, the partial derivatives are more positive than negative, so cost can be reduced by generally reducing the number of UAVs. This is an intuitive result which we knew already. Comparing the alternating gradient sizes in the bottom half of Figure 18 with the UAV numbers in the top suggests another way to cut the total cost: shave off the peaks and fill in the valleys of U_t. This is much like in the hanging chain problem where alternating spring forces on the links tend to smooth out kinks in the chain. Again, this is an intuitive result because it would clearly be costly to rapidly invest and divest in a given asset type.

Although there are 23 metrics in the military portfolio example, there are ultimately only two composite objectives to be optimized: costs fitness and performance fitness. These group fitnesses have the form of a weighted sum of the fitnesses of the individual metrics, where $f_i(\dots)$ is the fitness function for metric M_i:

$$f(M_1, M_2, \dots, M_{23}) = \sum_i w_i f_i(M_i)$$

The gradient of the group fitness can then be computed using chain rule where the gradients are taken with respect to all system counts. In the case of the military portfolio example, there are 140 system count variables.

$$\nabla f = \sum_i w_i \nabla f_i(M_i) = \sum_i w_i \frac{df_i}{dM_i} \nabla M_i$$

This is why we need to calculate the gradients of the individual metrics.

5.4.4 Gradient-Directed Mutations Implementation

To apply the gradient-directed mutations technique to the example military portfolio, the TMO model does not need to change at all: it's just a matter of replacing the external evaluator. As

before, the evaluator is called on each new member of the population to evaluate all the metrics. However, the evaluator is now also responsible for injecting new members into the population, much as the built-in mutation and crossover operators do in a basic GA. If the new member of the population is itself not an injection, one or more copies are made, modified, and injected into the population (note: the newly injected members are immediately returned to the evaluator for metric evaluation).

Each copy can be thought of as containing arrays of all the system counts over the 20 quarter timeframe. At the same time it is calculating the metrics, the evaluator calculates gradient arrays for the group fitness functions with respect to all the arrays of counts. Several different mutation subroutines were created to use the resulting gradient information to modify the copies. In essence, each subroutine takes in information much like that shown in Figure 18: an array of counts of any single system along with a corresponding array of partial derivatives. The direction of improvement should be indicated by positive derivatives since the objective functions are usually fitnesses. The subroutine then modifies the array of counts belonging to the copy.

The following mutation subroutines were developed. System count variables are left unchanged unless otherwise specified. Any counts that become negative as a result of mutation are truncated to zero:

- **BlockMutate**. This implements the "Block Cumulative" operator that was described in section 4.1.1 for the hanging chain problem.
- **RandomMutate**. This was mainly defined for testing purposes and ignores the gradients. It simply increments each count by 1, 0, or -1 with equal probability.
- **FitnessAsProbabilityMutate**. Each positive gradient component is divided by the largest positive component, and the result is interpreted as a probability. The analogous procedure is performed for negative components. Each count is incremented or decremented in the corresponding direction with the corresponding probability.
- **StratifiedMutate**. This is most similar to the "LP Norm" operator for the hanging chain problem. Here, the full range of the 20 fitness gradient components is divided into equal-sized strata, and each count is incremented, decremented, or left unchanged according to what stratum its corresponding component falls into. If the all components are positive, there are two strata, and the top stratum is incremented. The natural opposite procedure is followed if all components are negative. If the components span both positive and negative values, there are three strata, and the top stratum is incremented while bottom stratum decremented.

Any combination of system type, fitness function, and directed mutation subroutine can be applied to mutate each copy. Multiple combinations can be applied individually or in any sequence. Clearly, many other such mutation subroutines could be defined and tested to fully exploit the information contained in the gradients.

5.4.5 Gradient-Directed Mutations Genetic Algorithm Results and Discussion

Several attempts were made at using the mutation subroutines defined above to speed the GA evolution over the baseline, but the results achieved so far are incomplete. Note that the UAV-Vehicle-Soldier portfolio is a highly interdependent system of systems in the sense that at least a

few of all three type of systems must be present for the portfolio to perform effectively. Typically, progress towards a stable Pareto frontier seems to be faster at first relative to the basic GA implementation. Then, the cost and budget objectives seem to dominate the initial evolution with the result that some of the system counts are driven to zero. It then takes a long time to recover solutions which restore performance. What can also happen is that two of the three types of systems may have counts of zero in any given period at which point, the gradients of performance are zero. That means that the gradients cannot point the way to better portfolios. If only one of the types of system counts is zero, the performance will be zero but the gradient can still be non-zero because of the way the subgradients are chosen.

Although our results to date have been mixed, we have reasons to believe that gradient-directed mutation techniques will prove to be a potent way to speed up the evolution of holistic portfolio optimization problems using GAs. Most importantly, we have barely begun to explore the space of gradient-directed mutation operators and how they are combined and sequenced. We had a number of false starts before we found a way to get gradient-directed mutations to work well on the hanging chain problem. Moreover, due to the unexpected complexity of our military portfolio example, there may very well still be bugs in our implementation. Given the novelty of the approach, there is still room for improvement in how we develop these models. Developing good examples that can be used as templates would be one way to improve our process.

Another reason has to do with the purity of the gradient-directed mutations that we are injecting. Currently, the TMO implementation only performs directed mutation on top of the internal point mutation and crossover operators without the benefit of an intervening selection step. More often than not, the offspring generated by uncoordinated random mutation and crossover are less fit than the parents. This is especially true in the case of a portfolio problem where the space of bad decisions in the vicinity of a partially optimized design is so much larger than the space of good decisions. In contrast, directed mutations are specifically designed and coordinated to increase fitness. If they are applied after the usual mutation and crossover operations, they will frequently have to overcome a senseless disadvantage before they can begin to help. To the extent that directed mutations are just another source of genetic variation, they should be applied in parallel to point mutation and crossover.

Yet a third reason has to do with how we formulated the portfolio metrics. In particular, we frequently used functions such as $\min(\ldots)$ which have abrupt changes in behavior whereas in the real world, such changes may actually be smooth and gradual. We also did not consider multiple missions or other ways that systems provide value. For example, vehicles probably would contribute to a more comprehensive portfolio performance metric even if there were no UAVs. We could even relax the detailed causal analysis and instead formulate metrics in a more phenomenological way. One such formulation that we considered but did not pursue is based on homogeneous functions that are sometimes used in economics. This would be desirable if a metric were known to have a certain power law scaling with respect to overall portfolio size. For example, if it were known that the performance would double if all the system counts were doubled. Linear functions have this property, but so does a homogeneous function of the first degree:

$$F(U,V,S) = aU + bV + cS + dU^{\alpha}V^{1-\alpha} + eV^{\beta}S^{1-\beta} + fU^{\delta}V^{\gamma}S^{1-\delta-\gamma} + \cdots$$

The linear terms reflect the value of individual systems. The non-linear terms capture value interactions between systems, and interactions are important to consider when evaluating systems of systems. Homogenous functions of other degrees (p) could be used if there were diminishing returns ($p < 1$) or synergies between systems ($p > 1$):

$$G(U,V,S) = aU^p + bV^p + cS^p + dU^{\alpha}V^{p-\alpha} + eV^{\beta}S^{p-\beta} + fU^{\delta}V^{\gamma}S^{p-\delta-\gamma} + \cdots$$

Finally, note that even richer homogeneous functions can be defined through composition of homogeneous functions. Metric reformulations along any of the lines above could reduce the complexity of the gradient calculations. More importantly, they would help eliminate "flat" regions of the metric functions where the gradients don't provide any information about how to update the portfolio. This would likely improve the behavior of the gradient-directed mutation technique.

6 CONCLUSIONS

GAs provide powerful approaches to solving optimization problems that contain combinatorial decision variables and nonlinear or multiple objectives. Of course, there are still optimization problems which tax their abilities, holistic portfolio optimization being one. We are therefore interested in creating new ways to extend GA performance and applicability in challenging areas.

By analogy with nature, conventional GAs only generate population diversity through random genetic variation in the form of mutation and crossover. Better designs are then created via selection. However, adding new population members based on new genetic variation operators will almost always speed the optimization on a per-generation basis and frequently on a CPU-time basis. Moreover, random variation is not "intelligent" in any way and does not take advantage of problem structure or performance clues to direct better designs.

The general direction of better performance is suggested by gradients of objective functions with respect to numerical decision variables provided the objective function can be expressed as an analytic function of those variables. We developed the relatively simple "hanging chain" model as a research vehicle to explore this idea. Finding the shape of a hanging chain is a standard problem in the calculus of variations, and this model allowed us to tap into our physical intuition about force and energy to help develop various algorithms for using the gradients to generate effective mutations. By combining gradient information across multiple links, we were even able to coordinate variation in multiple count decision variables.

Some algorithms involved adding one or more parameters including a relaxation parameter. Tuning the relaxation parameter allows us to find effective steps sizes for moving the links in the chain. However, an interesting thing about optimizing over integers is that they define a natural scale to the size of the mutations; namely, one unit. Since the range of possible integer values in a portfolio problem is somewhat limited (typically a few hundred at most), we can use the minimum step and still reach the optimum in a reasonable number of generations.

By studying the hanging chain problem, we discovered a number of interesting things. For example, we learned that different genetic variation operators can be much more effective together than individually. In this sense, they are synergistic. Furthermore, we observed that genetic variation operators are more or less effective during different epochs of the optimization. The "every man for himself" method does a good job of removing large kinks in the first few generations. Then crossover helps combine parts of different chains that happen to be closer to the optimal shape. Eventually, the standard mutation operators run out of steam, and the gradient-directed mutation operators can fine tune the solution.

Measuring the speedup of one GA over another is not entirely straightforward. To begin with, time could be measured in a number of ways: the number of function evaluations, the number of generations, CPU time, or wall clock time. Then, the speedup might be defined in terms of a ratio of times for a certain level of performance or a ratio of performance distance from optimal given a certain amount of time. It's not always possible to do the latter since the true optimum is not generally known. Ultimately, we measured the wall clock time it took for the slower algorithm to reach a certain point and took the ratio with how long it took for the faster algorithm to achieve the same or better results. In the case of multi-objective optimization, it's not totally unambiguous when one Pareto frontier becomes better than another, so we had to define somewhat arbitrary rules (albeit conservative rules). Finally, the progress of the baseline conventional GAs virtually stops beyond some point which makes the gradient-directed speedup appear to increase the more baseline generations are run. Because of this, we can run the

baseline single-objective hanging chain problem for a week and claim a speedup of million or more. In the case of the multiobjective hanging chain, we can claim speedups of at least a thousand.

In the case of multiobjective optimization, we found that gradient-directed mutations not only sped convergence, they also produced qualitatively better results. In particular, the set of non-dominated solutions that was obtained had a greater spread over the space of metrics. This is important because it gives decision-makers the fuller picture of the possible trade space.

In order to develop our methods further, we developed a notional example of a contemporary military portfolio along with its defined mission. The example adds complexity in the form of combinatorial decision variables, dependency constraints between those variables, and more types of platforms. To simplify the interpretation of the trade space, it also demonstrates the combination of multiple metrics into fitness functions to be maximized in contrast to energy functions to be minimized. The example tells a good story while illustrating a number of modeling and analysis techniques which should prove useful to others who want to develop holistic portfolio optimization models.

We discovered that defining holistic portfolio metrics for the example military portfolio was surprisingly difficult. Conventional sequential portfolio optimization involves defining metrics for individual systems and then defining another set of metrics for the portfolio as a whole given the optimized designs of the individual systems. In both cases, formulating metrics is a non-trivial task. However, in order to realize the benefits of holistic portfolio optimization, metrics must be defined which capture the system-of-systems interactions between the design of the individual systems and the counts of those systems over time. This makes defining good metrics much harder still. Doing it right requires careful consideration of the CONOPS of one or more missions.

One of the things that make GAs attractive is that they are relatively easy to implement. Part of the price of speeding them up using gradient-directed mutations is that the implementation becomes much harder and the evaluation of each new design is slowed. Once we had analytic formulas for our example military portfolio metrics, implementing them was not hard, but it did become rather complicated to interleave the calculation of their derivatives. Some of the difficulty stemmed from the need to keep track of the multitude of cases and the subgradient calculations between cases. However, the main complicating factor was the complex way the insurgency metric depended on the decision variables: the number of insurgent cells depends implicitly on the earlier numbers of cells, not just explicitly on the decision variables. This necessitated an iterative application of the chain rule. In the end, the gradient calculations increased the evaluation time by roughly a factor of ten. Better tools are needed to make the calculation of derivatives simpler and more efficient. We believe that automatic differentiation may offer a good approach.

We have demonstrated the utility of gradient-directed mutations for holistic portfolio optimization over time where both the system technologies and the counts of systems are decision variables. While HPO is still proving difficult when combinatorial variables are included, we are still at an early stage in the research. With further development, gradient-directed mutations may even help in traditional combinatorial optimization problems. Ultimately, we believe that there is great promise in taking advantage of problem structure to give hints to GAs in the form of intelligently informed mutations. Success in this effort would open up new approaches to operations research problems more generally.

7 REFERENCES

1. Salomon, R., *Evolutionary Algorithms and Gradient Search: Similarities and Differences*, IEEE Transactions on Evolutionary Computation 2, 45-55, 1998.
2. Bhandari, D., Pal, N.R., and Pal, S.K., *Directed Mutation in Genetic Algorithms*, Information Sciences 79, 251 – 270, 1994.
3. Guan, J. and Aral, M.M., *Progressive Genetic Algorithm for Solution of Optimization Problems with Nonlinear Equality and Inequality Constraints*, Applied Mathematical Modeling 23, 329–343, 1999.
4. Yi, N., Dechun, T., and Yuzheng, L., *Genetic Algorithm Diagnosis of Individual Cell Frequencies in a Coupled Cavity Chain*, Nuclear Instruments and Methods in Physics Research A 462, 356–363, 2001.
5. Chen, C., Chen, Y., and Zhang, Q., *Enhancing MOEA/D with Guided Mutation and Priority Update Multiobjective Optimization*, IEEE Congress on Evolutionary Computation, 209-216, 2009.
6. Tang, P. and Tseng, M., *Adaptive Directed Mutation for Real-Coded Genetic Algorithms*, Applied Soft Computing 13, 600–614, 2013.
7. Heitzinger, C, and Selberherr, S., *An Extensible TCAD Optimization Framework Combining Gradient Based and Genetic Optimizers*, Microelectronics Journal 33, 61-68, 2002.
8. Smith, A.E. and Coit, D.W., *Constraint Handling Techniques – Penalty Functions*, Handbook of Evolutionary Computation, C5.2-1 – C5.2-6, 1997.
9. Michalewicz, Z. and Janikow, C.Z., *GENOCOP: A Genetic Algorithm for Numerical Optimization with Linear Constraints*, Communications of the Association for Computing Machinery 39, Article 175, 1996.
10. Farmani, R. and Wright, J.A., *Self-Adaptive Fitness Formulation for Constrained Optimization*, IEEE Transactions on Evolutionary Computation 7, 445-455, 2003.
11. Mezura-Montes, E. and Coello, C.A., *Constraint-handling in Nature-inspired Numerical Optimization: Past, Present, and Future*, Swarm and Evolutionary Computation 1, 173-194, 2011.
12. Hoffman, M.J., Henry, S.M., Thompson, B.M, *Whole System Trades Analysis Tool (WSTAT) Overview*, SAND2014-20604PE, Sandia National Laboratories, Albuquerque, NM, 2014.
13. Davis, S.J. et. al., *Maximizing the U.S. Army's Future Contribution to Global Security using the Capability Portfolio Analysis Tool (CPAT)*, Interfaces 46, in press, 2016.
14. Eddy, J.P., *JEGA - A Tool for Multi-objective Optimization*, SAND2006-6659C, Sandia National Laboratories, Albuquerque, NM, 2006.
15. Eddy, J.P, *Technology Management Optimization (TMO)*, SAND2010-2622C, Sandia National Laboratories, Albuquerque, NM, 2010.
16. Center for Systems Reliability, *Technology Management Optimization Fact Sheet*, http://www.sandia.gov/CSR/tools/tmo.html, Sandia National Laboratories, Albuquerque, NM, 2016.

APPENDIX A: MAX BLOCK CUMULATIVE FORCE ALGORITHM

In the description of the Block Cumulative Directed Mutation in section 4.1.1, we claim to have an efficient method (scaling linearly with the number of links n) for finding the set of contiguous links (i.e., a block) having maximal force in the x and y directions. For the Block Cumulative method, this block of links is then moved in the direction indicated by the cumulative force on that block – providing a means to coordinate many link updates simultaneously. In this Appendix, we document the method by which we capture this block.

Suppose we have already calculated the net force on each link and now wish to find the block of links with the maximal cumulative (absolute value) force in the y direction (this method would proceed analogously for the block of links having maximal x-force). To do this, we form a new vector that tracks the cumulative y-force at each link from left to right across the chain. Here, the cumulative y-force on the i^{th} link is the sum of the y forces on and to the left of link i. Next, we find the two positions along the chain having the largest and smallest cumulative y-force (call these links i_{max} and i_{min}, respectively). If there is a tie in the extremes (minimum or maximum) cumulative force, any of the selected links could be chosen. Finally, if $i_{min} < i_{max}$ then the set of links having maximum y-force is given by $B_y = \{i_{min} + 1, i_{min} + 2, \dots, i_{max}\}$. Otherwise if $i_{max} < i_{min}$ then the set of links is $B_y = \{i_{max} + 1, i_{max} + 2, \dots, i_{min}\}$. In essence we chose the set of links starting one to the right of the leftmost extreme and continuing to the rightmost extreme.

Since the procedure consists only of calculating a cumulative force vector of size n and finding the minimum and maximum in this vector, the overall complexity of the procedure is $O(n)$. To see this more clearly, consider Figure 19 below which plots and example y force on each link of a MECE chain with n=8. The left-to-right cumulative y force on each link is displayed underneath the system, and the minimum and maximum cumulative y forces are highlighted in red. Thus, the contiguous block with the largest cumulative force occurs from link 2 to link 6.

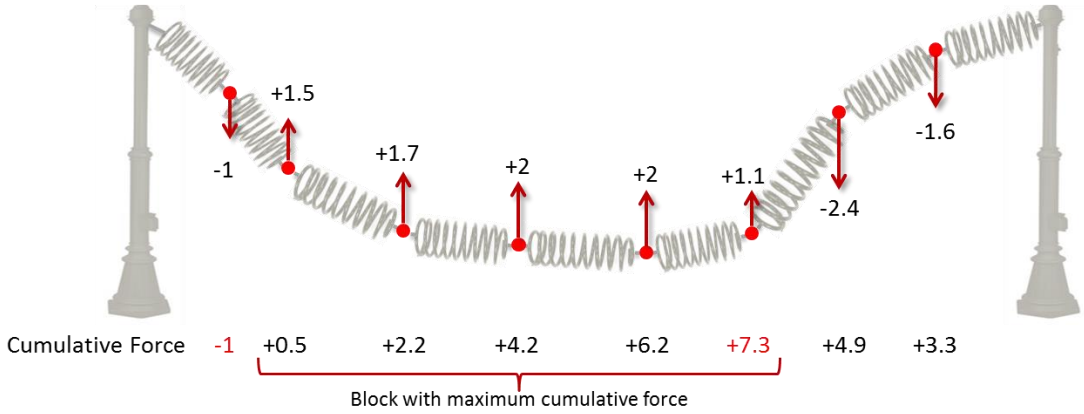


Figure 19. A MECE system with forces in the y coordinate. The cumulative force from left to right demonstrates how to find the block of links with maximal force.

APPENDIX B: BUDGET, COST, AND GRADIENT CALCULATIONS

Since the portfolio metrics are defined piecewise, there are many cases to evaluate, and it's frequently simpler to express the calculations in program code form rather than in formulae. The external evaluator is written in VisualBasic.NET, and the figures below show the code for each quarter inside a loop indexed by quarter `t`. Partial derivatives of any metric with respect to a system count (at constant values of the other 139 system counts) are represented by identifiers with an underscore (_) in the name. The identifier prefix `B` means budget in the current quarter, `C` means the total cost over all quarters, and `U` means the count of original UAVs. So the notations "`B_U(0, t)`" and "`C_U(t)`" respectively denote the partial derivatives of `B` and `C` with respect to `U` in the current ("`0`") quarter, `t`. It also happens that the system counts in the previous ("`1`") quarter affect the budget in the current quarter. So the notation "`B_U(1, t)`" denotes the derivative of `B` with respect to `U` in the *previous* quarter, `t-1`.

Figure 20 contains the code for calculating contributions to the budget from the counts of *all* UAV platforms along with the partial derivatives of the budget with respect to the count of *original* UAVs. Since the platforms are the same for original and upgraded UAVs, any increase in the total number of UAVs means the budget in this quarter will increase in proportion to the UAV platform purchase price. Decreases in the total count of UAVs reduce the quarterly costs, but since there are no proceeds from divestment, the purchase price is not recovered. This implies a "kink" in the budget due to purchases, so we average the partial derivatives across the kink using a "subgradient factor" of one-half, corresponding to the midpoint of the subgradient at the kink. The quarterly costs don't have a kink, so they add a constant to the partial derivative.

```
budget(t) += QuarterlyCost("UAV") * numOriginalUAVs(t)
budget(t) += QuarterlyCost("UAV") * numUpgradedUAVs(t)
Dim deltaTotalUAVs As Double = numOriginalUAVs(t) + numUpgradedUAVs(t) _
                             - numOriginalUAVs(t - 1) - numUpgradedUAVs(t - 1)
If deltaTotalUAVs >= 0 Then
    budget(t) += Price("UAV") * deltaTotalUAVs

    Dim subGradientFactor As Double = If(deltaTotalUAVs = 0, 0.5, 1)
    B_U(0, t) += subGradientFactor * Price("UAV")
    B_U(1, t) -= subGradientFactor * Price("UAV")
End If
B_U(0, t) += QuarterlyCost("UAV")
```

Figure 20. UAV platform contributions to the budgets and their gradients.

Figure 21 contains the code for calculating contributions to the budget from the original UAV sensors. Note that the original UAV counts are used as the original sensor counts since they are the same. As before, only increases in the counts increase the budget due to purchases while quarterly costs are incurred in any case. The sensor price and quarterly cost contribute to the partial derivatives in direct analogy with the platform purchase and quarterly costs.

```
budget(t) += QuarterlyCost("Sensor 0") * numOriginalUAVs(t)
Dim deltaOriginalUAVs As Double = numOriginalUAVs(t) - numOriginalUAVs(t - 1)
If deltaOriginalUAVs >= 0 Then
    budget(t) += Price("Sensor 0") * deltaOriginalUAVs

    Dim subGradientFactor As Double = If(deltaOriginalUAVs = 0, 0.5, 1)
    B_U(0, t) += subGradientFactor * Price("Sensor 0")
    B_U(1, t) -= subGradientFactor * Price("Sensor 0")
End If
B_U(0, t) += QuarterlyCost("Sensor 0")
```

Figure 21. Original sensor contributions to the budgets and their gradients.

After all the budgets and budget gradients are calculated, then the total cost and partial derivatives of total cost with respect to `U` in each period can be calculated as shown in Figure 22. Obviously, the partial derivative of cost with respect to `U` in the current quarter depends on the partial derivative of `B` in the current quarter with respect to `U` in the current quarter. But less obviously, it also depends on the partial derivative of `B` in the *next* quarter with respect to `U` in the current quarter (unless it's the last quarter in which case there isn't a next quarter).

```
cost += budget(t)

C_U(t) = B_U(0, t)
If t <> 20 Then
    C_U(t) += B_U(1, t + 1)
End If
```

Figure 22. Example total cost and total cost gradient calculations.

DISTRIBUTION

1	MS0359	D. Chavez, LDRD Office	1911 (electronic copy)
1	MS1188	Alan Nanco	6114 (electronic copy)
1	MS1188	Bruce M. Thompson	6133 (electronic copy)
1	MS1188	Mark A. Smith	6133 (electronic copy)
1	MS1188	Stephen M. Henry	6133 (electronic copy)
1	MS1188	John P. Eddy	6133 (electronic copy)
1	MS1188	Craig R. Lawton	6135 (electronic copy)
1	MS0899	Technical Library	9536 (electronic copy)


Sandia National Laboratories