AN EVOLUTIONARY OPTIMIZATION SYSTEM FOR SPACECRAFT DESIGN

Alex S. Fukunaga and Andre D. Stechert
Jet Propulsion Laboratory, MS 525-3660
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
Email: {alex.fukunaga,andre.stechert}@jpl.nasa.gov

Abstract

Spacecraft design optimization is a domain that can benefit from the application of optimization algorithms such as genetic algorithms. However, there are a number of practical issues that make the application of these algorithms to real-world spacecraft design optimization problems difficult in practice. In this paper, we describe DEVO, an evolutionary optimization system that addresses these issues and provides a tool that can be applied to a number of real-world spacecraft design applications. We describe two current applications of DEVO: physical design of a Mars Microprobe Soil Penetrator, and system configuration optimization for a Neptune Orbiter.

1 Introduction

In theory, many aspects of spacecraft design can be viewed as constrained optimization problems. Given a set of decision variables X and a set of constraints C on X, constrained optimization is the problem of assigning values to X that minimize or maximize an objective function F defined on X subject to the constraints C. In practice, there are a number of theoretical and practical obstacles that make constrained optimization of spacecraft designs difficult.

First, while optimization of smooth, convex objective functions is well understood (and efficient algorithms are known to exist, see, for example, [Fle87]), global optimization on surfaces with many local optima is not. Traditional approaches to optimization usually fare poorly on these so-called "rugged" surfaces, which are often characteristic of real-world optimization problems.

Second, many real-world optimization problems are *black-box optimization problems*, in which the structure of the cost function is opaque. That is, it is not feasible to analyze the cost surface by analytic means in order to guide an optimization algorithm. Often, this occurs when $F(X)$ is computed by a complex simulation about which the optimization algorithm has no information (e.g., to evaluate a candidate spacecraft design, we could simulate its operations using a suite of legacy FORTRAN code about which very little is known except for its I/O specifications). Black-box optimization problems of this kind are challenging from a practical point of view for two reasons: 1) Executing a black-box simulation in order to evaluate a candidate solution is usually very expensive relative to, for example, evaluating a cost function that is expressed as a system of equations, and can take on the order of several *minutes*. This is particularly problematic because optimization algorithms for black-box problems are necessarily *blind* search algorithms that must repeatedly choose sample points from the solution space, evaluate them by running the simulation, and then apply various heuristics in order to choose the next points to sample, 2) Interfacing optimization tools to black-box simulations can be difficult, particularly when the black box is a complex software system that involves various components written in different languages, possibly running on a distributed environment running on a number of different platforms.

Finally, many spacecraft design engineers do not have the optimization expertise required to apply state of the art algorithms to their problems. Obtaining this expertise is often prohibitively expensive; as a consequence, optimization using algorithmic methods is sometimes not even attempted, because of the perception that it is not worth the effort and expense.

In this paper, we describe Design Evolver (**DEVO**), an optimization system developed in an effort to address these issues. To address the problem of optimization of difficult cost surfaces, DEVO implements a generic, reconfigurable implementation of an evolutionary optimization algorithm. To overcome the practical difficulties described above that arise when designing tools for black-box optimization problems, DEVO is integrated with MIDAS [GPS95], a recently developed integrated spacecraft design environment, making it possible to apply automated optimization to any spacecraft design model specified in this environment.

The rest of this paper is organized as follows. Section 2 describes the architecture of the DEVO system, focusing on the practical issues that arise in the integration of an evolutionary optimization algorithm into a spacecraft design environment, and the automated reconfiguration of the optimization algorithm for a particular problem instance. In Section 3, we describe two spacecraft design optimization problems which are currently being used as testbed applications for DEVO: the NASA New Millennium DS-2 Mars Microprobe, and the Neptune Orbiter spacecraft.

2 The Design Evolver (DEVO) System

DEVO is a system for spacecraft design optimization currently being developed at the Jet Propulsion Laboratory (JPL). The goal of DEVO is to provide an optimization tool that is seamlessly integrated into an existing computer-aided design (CAD) environment for spacecraft, which enables users to apply optimization algorithms, including evolutionary algorithms, with a minimal amount of human effort. A fundamental assumption in the DEVO design is that CPU cycles are plentiful and cheap relative to the cost of an engineer hand-tuning an optimization engine. In this section, we describe the DEVO system. We first describe the evolutionary optimization algorithms implemented in DEVO. We then describe MIDAS, the design environment into which DEVO has been integrated. Then, we describe various features of DEVO that address the practical issues that arise in the application of evolutionary optimization techniques to real-world, spacecraft design problems.

2.1 Reconfigurable Evolutionary Algorithms

The central component of DEVO is the Reconfigurable Evolutionary Algorithm (**REAL**). The REAL is an implementation of a generic evolutionary optimization strategy that can be reconfigured at runtime to behave as one of the various classes of evolutionary algorithms. Figure 1 (following [BS96]) shows the general schema for evolutionary algorithms which the REAL implements.

Briefly, an evolutionary algorithm works as follows: a population of sample points from the cost surface is generated. In a process analogous to biological evolution, this population is evolved by repeatedly selecting (based on relative optimality) members of the population for reproduction, and recombining/mutating to generate a new population.

By providing different implementations of functions such as *initialize*, *recombine*, *mutate*, and *select*, and a selection of encodings (representations) of solutions (e.g., bit-string encoding, possibly with Gray coding, floating point representations, etc.) that can be chosen at runtime, it is possible to reconfigure the REAL to simulate a wide variety of evolutionary algorithms. For example, using a null *recombine* function and implementing a *mutate* function that applies Gaussian mutation, we achieve the canonical Evolutionary Programming (cf., [Fog95]) algorithm.

```
$t := 0$
**initialize** $P(t)$;
**evaluate** $P(t)$;
**while not terminate do**
  $P'(t) :=$ **recombine** $P(t)$;
  $P''(t) :=$ **mutate** $P'(t)$;
  **evaluate** $P(t)$;
  $P(t+1) :=$ **select** $(P''(t) \cup Q)$;
  $t := t + 1$;
**end while**
```

Figure 1: Algorithm schema for an evolutionary algorithm P is a population of candidate solutions; Q is a special set of individuals that has to be considered for selection, e.g., $Q = P(t)$.

Currently, the REAL supports bit-string representations of numerical parameters, as well as floating point number representations. Various mutation, recombination, and selection operators are available. Furthermore, the REAL supports a number of different population structures, including the traditional generational population structure [Gol89], a steady-state population structure [Sys89], and a distributed population structure [Tan89]. Thus, the REAL can be configured to simulate a wide range of common Genetic Algorithm (cf. [Gol89]) and Evolutionary Programming (cf. [Fog95]) variants.

2.2 Spacecraft Design Model

A *spacecraft design model* is a software simulation of a spacecraft design. The design model takes as input decision variables to be optimized, and outputs an objective function value, which is assigned as the result of an arbitrarily complex computation (i.e., the simulator is a black-box simulation).

Thus, the design model is domain-specific, and is provided by the end users, i.e., spacecraft designers. In order for an optimization tool such as DEVO to be useful in practice, it must support a wide range of design models, which may consist of models implemented using various languages on different platforms. It is not feasible to expect spacecraft designers to implement their models in a particular language on a particular platform – if such inconvenient constraints were imposed, the optimization system will not be used by spacecraft designers.

The Multidisciplinary Integrated Design Assistant for Spacecraft (**MIDAS**) [GPS95] is a computer-aided design environment developed at JPL that allows a user to integrate a system of (possibly distributed) design model components

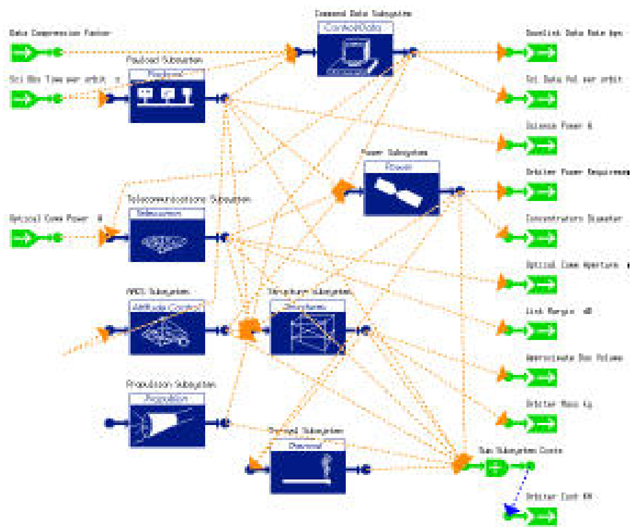

Figure 2: Screen shot of a MIDAS methogram (part of the Neptune Orbiter model).

using a *methogram*, a graphical diagram depicting the data flow of the system. Each node in the methogram corresponds to a design model component, which may be one of 1) a model in a commercial design tool such as IDEAS, NASTRAN, or SPICE, 2) a program written in C, C++, or FORTRAN, 3) a MIDAS built-in tool, or 4) an embedded methogram (i.e., methograms can be hierarchical). Inputs to nodes in the methogram correspond to input parameters for the component represented by the node, and outputs from a methogram node correspond to output values computed by the component. Since it was implemented as a distributed object system and since an output node can be used to compute an arbitrary function of the parameters in the model, MIDAS provides a uniform interface to a wide variety of design models without requiring optimization algorithms to have strong dependencies on the target simulation. Figure 2 shows a screen shot of the MIDAS methogram for the Neptune Orbiter model.

2.3 Automated Configuration of the REAL

As we mentioned in Section 1, a significant practical obstacle to applying optimization algorithms to spacecraft design is the optimization knowledge and effort required of spacecraft designers. We therefore designed DEVO to be usable with as little input as possible beyond what is already provided by the user in the MIDAS spacecraft design model.

In order to run DEVO, the user is required to input the following:

- A MIDAS methogram that encapsulates the design model;
- A list of decision variables, as well as ranges of their possible values. These may be continuous, discrete, or enumerated types;[1]
- An output from a methogram node that corresponds to the user's objective function value; and
- A termination condition for a run of the optimization algorithm. This can be either 1) a time limit, 2) a maximum number of simulation runs, or 3) a simple check for convergence of the algorithm, i.e., no improvement is made for some number of simulation runs (a default value is provided).

The user input listed above is sufficient for DEVO to automatically configure the REAL to an appropriate default configuration and run the evolutionary algorithm. Based on the decision variable types and ranges specified by the user, the genome evolved by the REAL is appropriately configured, and the evolutionary algorithm is executed. A user can, of course, manually configure the REAL using either a command-line or graphical user interface.

Thus, the effort and knowledge required by the spacecraft designer to run DEVO is minimal, since essentially all that is required for a user to use DEVO to optimize a design is to specify the decision variables and the constraints on them, and to specify an objective function.[2]

We should make it clear that by no means are we claiming that DEVO can provide a default configuration that works well for all spacecraft design problems. In fact, in the absence of any knowledge of the cost surface structure, it is quite possible that any default configuration of the REAL may be no better than random search.[3] *However, even if the default configuration chosen by DEVO performs relatively poorly, we argue that applying some optimization algorithm is better than not applying any optimization at all* (particularly when computational resources are readily available).

When a run of a particular configuration has terminated, DEVO continues the optimization process by saving the best solution found so far, and restarting the optimization process using another configuration.[4] This process is repeated until terminated by the user. The strategy currently used by DEVO to choose the next REAL configuration is a simple randomized strategy: generate the next configuration randomly, the only constraint being that the configuration is compatible with the decision variable types.[5] The general problem of *adaptive problem solving*, i.e., reconfiguring a

[1]Enumerated types are mapped onto discrete values by DEVO.

[2]In many cases, this objective is already available in the methogram.

[3]See [WM94] for a recent theoretical perspective on this issue.

[4]We are currently investigating whether it is better to start the next optimization run from scratch, or to seed the initial population using solutions found in previous runs.

[5]It is acceptable to repeat configurations, since the performance of evolutionary algorithms is stochastic.

problem solver to perform well on a particular problem instance, poses many difficult theoretical challenges, and is beyond the scope of this paper (see [FCM$^+$97] for a more detailed discussion of this problem, as well as more sophisticated solutions that we are currently investigating).

2.4 Optimization as an Interactive Process

Most of the work on evolutionary optimization focuses on optimization as a fully automated process, in which the *initialize* step in Figure 1 is accomplished by random initialization. Design optimization, however, is often an interactive process, since the designers who developed the model have sufficient domain expertise to suggest some "reasonable guesses" as to what good decision variable parameters might be. Indeed, in many cases, it is difficult for a completely automated optimization process to produce designs of better quality than a human engineer.

A reasonable alternative is to attempt *local optimization* of a design that is initially specified by a human engineer. Therefore, in addition to the usual random initialization functions, we have implemented an initialization function for the REAL that generates the initial population based on random perturbations of decision variable parameters which the user has specified in the model. The perturbations apply random noise with Gaussian or uniform noise applied at the user's discretion.

2.5 Additional Implementation Details

2.5.1 Handling Instabilities in Black-Box Simulation

A common problem when trying to apply evolutionary algorithms to black-box simulations is the possibility of instability in the simulations. Spacecraft design models are often one-of-a-kind prototype systems, designed and implemented to provide proof of concept in the hands of an experienced engineer. Consequently, they are not necessarily robust enough to be executed with the thousands of different assignments of decision variable values that an evolutionary algorithm attempts. A particular input parameter combination could, for example, cause an arithmetic exception, causing the simulation to crash. If the optimization system is not designed to anticipate such failures, the whole optimization system could fail as a result. It could be argued that such instability could be symptomatic of an unreliable simulator whose results should not be trusted at all (and therefore needs to be fixed immediately). However, we take the view that it is not feasible to demand that the simulation software be made completely robust for the sake of the optimization system and designed DEVO so that it would circumvent simulation instability as much as possible.

DEVO protects the optimization system from simulation software instability by exploiting fault detection features built into MIDAS, and by separating the optimization process from the simulation process as much as possible. MIDAS is capable of detecting common failures (e.g., core dumps, arithmetic exceptions) that occur when executing a process that corresponds to a node in a methogram. DEVO monitors this information, and immediately aborts the execution of the simulation upon detecting a failure. Some failures can actually cause MIDAS itself to crash. Since DEVO is implemented as a separate process which invokes MIDAS and manipulates it through its CORBA interface, DEVO can detect MIDAS crashes and abandon the candidate solution evaluation that called the failed MIDAS simulation. It is, of course, not possible to detect with certainty if a simulation has entered an infinite loop; however, if DEVO has been waiting for the result of a simulation for an abnormally long time (e.g., if a simulation run is taking n standard deviations more time than an average simulation run to date), then it is assumed that the simulation is trapped in a loop, and DEVO will terminate the simulation.

An interesting issue is what to do with the evaluation of solutions that cause failures that are detected as described above. We currently apply the simple policy of assigning the worst possible fitness values to these solutions. On one hand, this has the effect of causing the evolutionary algorithm to avoid solutions that are very similar to the offending solutions. Assuming that the simulation is unstable in *regions* of the solution space, rather than isolated points, this is a reasonable policy. On the other hand, if this is not the case, then this policy could cause an undesirable bias in the evolutionary search. We are currently investigating the significance of this bias.

2.5.2 Parallelization

Executing a complex spacecraft design model simulation often takes a significant amount of time. For example, a single execution of our current Neptune Orbiter (see 3.2) takes several minutes on a Sun Ultra workstation. Given that a single run of an evolutionary algorithm requires hundreds to thousands of candidate solution evaluations, this poses a serious problem, if we want to complete the optimization processes in a reasonable amount of time. Fortunately, evolutionary algorithms are particularly well-suited to parallelization, since each candidate solution can be evaluated independently of the other solutions, i.e., the *evaluate* step in Figure 1 can be parallelized with near-linear efficiency. Therefore, DEVO distributes simulations over a network of workstations using Parallel Virtual Machines (PVM) [GBD$^+$94].

3 Spacecraft Optimization Problems

In this section, we describe two specific spacecraft design optimization problems to which we are currently applying the DEVO system. The first is a low-level optimization of the physical dimensions of a soil penetrator microprobe. The second is a system-level optimization of the configuration of the communication system of an orbiter spacecraft. These examples are illustrative of the wide range of different spacecraft design optimization problems to which DEVO can be applied.

3.1 The Mars Soil Penetrator Microprobe

As part of the NASA New Millennium program, two microprobes, each consisting of a very low-mass aeroshell and penetrator system, are planned to launch in January, 1999 (attached to the Mars Surveyor lander), to arrive at Mars in December, 1999. The 3kg probes will enter the Martian atmosphere and orient themselves to meet heating and impact requirements. Upon impacting the Martian surface, the probes will punch through the entry aeroshell and separate into fore- and aftbody systems. The forebody will reach a depth of 0.5 to 2 meters, while the aftbody will remain on the surface for communications.

Each penetrator system includes a suite of highly miniaturized components needed for future micro-penetrator networks: ultra low temperature batteries, power microelectronics, and advanced microcontroller, a microtelecommunications system and a science payload package (a micro-laser system for detecting subsurface water).

The optimization of physical design parameters for a soil penetrator based on these Mars microprobes is the first testbed for the DEVO system. The microprobe optimization domain in its entirety is very complex, involving three stages of simulation: separation from the Mars Surveyor, aerodynamical simulation, and soil impact and penetration. The complete design model for the penetrator is currently under development. Below, we describe the current model, which implements the simulation of stage 3 (impact/penetration).

Given a distribution on parameters describing the initial conditions including the angle of attack of the penetrator, the impact velocity, and the hardness of the target surface, the optimization problem is to select the total length and outer diameter of the penetrator, so as to maximize the expected ratio of the depth of penetration to the length of the penetrator. We maximize this ratio, rather than simply maximizing the depth of penetration, since for the Mars microprobe science mission, the depth of penetration should be at least as large as the overall length of the penetrator).

Using the default REAL configuration generated by DEVO (a canonical generational GA using bit-string encodings, one-point crossover, bit-flip mutation, population size of 50), it is consistently possible to generate a near-optimal design after about 50 generations.

3.2 The Neptune Orbiter

Neptune Orbiter is a mission concept currently being studied under the Outer Planet Orbital Express program at the Jet Propulsion Laboratory. The goals of the mission are to put a spacecraft in orbit around Neptune using state-of-the-art technologies in the areas of telecommunications, propulsion, orbit insertion, and autonomous operations. The spacecraft is expected to arrive at Neptune (30 a.u.) 5 years after launch in 2005 using a Delta launch vehicle. The subsystem requirements include 100 kbps data rate, solar electric propulsion, solar concentrator power source and a cost of less than $400 million (in FY 94 dollars).

For the initial phase of the optimization effort, the focus is on the orbital operations of Neptune Orbiter. The launch and cruise phases of the mission will be included in the optimization once the orbiter problem is well understood. The driving constraints of the orbiter problem are the optical communication aperture, transmit power and spacecraft mass. The transmit power is a direct input into the integrated spacecraft design model. The other inputs include the science observation time per orbit and the data compression factor. The output of the model that is being maximized is the science data volume per orbit. For designs in which the spacecraft mass is greater than 260 Kg, the data volume output is zero. A spacecraft with a dry mass of greater than 260 Kg is too heavy to lift on the target launch vehicle. Thus the mass limit constrains the optimization problem. Currently, we are using cost models in conjunction with the simulation of the orbiter as described above to obtain our cost function - a quantitative estimate of the science return (measured in, e.g., volume of science data obtained per dollar cost of the spacecraft).

4 Conclusions

Designing a widely applicable tool for black-box design optimization poses a significant technical challenge. In this paper, we have described DEVO, an evolutionary optimization system for spacecraft design that provides a design optimization tool that can be applied to real-world spacecraft design optimization problems with minimal human effort.

Much of the recent work in the evolutionary algorithm literature focuses on development of specialized representations and techniques to customize an evolutionary algorithm for a particular application. While we agree that developing specialized algorithms for particular applications is the best methodology for obtaining the best performance for any particular domain, this approach is often infeasible in practice, due to the human expertise and effort required to develop a specialized algorithm.

For problems that are unique in nature, a promising approach is to provide tools that make it possible to apply very general methods with little overhead. As long as the application of the method is virtually free of human effort, it is worthwhile to use available computational resources to approach the problem in a "brute-force" manner off-line, since the potential benefits of improving design quality can be quite substantial. The development of DEVO is a first step in this direction.

So far, we have found that the default behavior of DEVO has been sufficient for finding near-optimal solutions to the Mars soil penetrator microprobe problem.[6] However, we believe that in order for reconfigurable systems such as DEVO to become more useful, techniques for intelligently, automatically configuring the system to suit a particular problem instance must be developed and integrated into the system. We are currently investigating this problem [FCM+97].

Finally, we note the utility of system development efforts such as DEVO to the evolutionary algorithm research community. A myriad of promising approaches to evolutionary optimization have been proposed in the literature. However, the success of a particular technique for a given problem depends largely on the match between the technique and the problem [WM94], and thus, assessing the utility of a particular approach is mostly an empirical, problem-specific issue. Since applying a new technique to a real-world problem is often difficult and time-consuming, evolutionary algorithm researchers often restrict their evaluation of new approaches to synthetic cost functions (e.g., [DeJ75], [WMRD95]) or other easily implemented problems (e.g., the Traveling Salesperson Problem) whose relationship to most real-world problems is tenuous. By providing a stable, uniform interface to a wide variety of black-box optimization problems, DEVO provides evolutionary algorithm researchers with a framework into which many new techniques can be easily integrated, enabling the evaluation of new approaches on real-world problems. As an example, we have recently integrated an *incremental evolution* technique [FK95] into DEVO, and demonstrated its utility (compared to standard genetic algorithms) on the DS-2 probe design problem (see [FCM+97] for details).

Acknowledgments

The research described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Thanks to Julia Dunphy and Jose Salcedo for assistance with MIDAS. Bob Glaser and Celeste Satter provided the domain models for the Mars microprobe and Neptune Orbiter, respectively.

[6]The randomized reconfiguration described in 2.1 has not been necessary so far; this is due to the fact that the default configurations of the REAL were calibrated using the Mars microprobe and Neptune Orbiter domains (the first problems to which the REAL has been applied). The default configurations may not be appropriate for future problem instances.

References

[BS96] T. Back and H-P Schwefel. Evolutionary computation: An overview. In *Proc. IEEE International Conf. Evolutionary Computation*, 1996.

[DeJ75] K. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Department of Computer and Communication Sciences, Ann Arbor, Michigan, 1975.

[FCM+97] A.S. Fukunaga, S. Chien, D. Mutz, R. Sherwood, and A. Stechert. Automating the process of optimization in spacecraft design. In *Proc. IEEE Aerospace Conf. (to appear)*, 1997.

[FK95] A.S. Fukunaga and A.B. Kahng. Improving the performance of evolutionary optimization by dynamically scaling the evaluation function. In *Proc. IEEE International Conf. on Evolutionary Computation (ICEC)*, 1995.

[Fle87] R. Fletcher. *Practical methods of optimization, 2nd ed.* Wiley, 1987.

[Fog95] D.B. Fogel. *Evolutionary computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - a user's guide and tutorial for networked parallel computing*. MIT Press, 1994.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[GPS95] J. George, J. Peterson, and S. Southard. Multidisciplinary integrated design assistant for spacecraft (midas). In *Proceedings of American Institute of Aeronautics and Astronautics (AIAA)*, 1995.

[Sys89] G. Syswerda. Uniform crossover in genetic algorithms. In *Proc. International Conf. on Genetic Algorithms (ICGA)*, 1989.

[Tan89] R. Tanese. Distributed genetic algorithms. In *Proc. International Conf. on Genetic Algorithms (ICGA)*, 1989.

[WM94] D.H. Wolpert and W.G. Macready. The mathematics of search. SFI-TR-95-02-010, 1994.

[WMRD95] D. Whitley, K. Mathias, S. Rana, and J. Dsubera. Building better test functions. In *Proc. International Conf. on Genetic Algorithms (ICGA)*, 1995.