Application of an Incremental Evolution Technique to Spacecraft Design Optimization

Alex S. Fukunaga
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, MS 525-3660
Pasadena, CA 91109-8099
alex.fukunaga@jpl.nasa.gov

Abstract

Based on the intuition that it is often easier to learn to solve difficult problems after similar, simpler problems have been learned, incremental evolution is a recently proposed extension to evolutionary algorithms in which the evaluation function against which a population is evolved is *scaled* over time. This paper presents an application of incremental evolution to the problem of physical design optimization of a Mars microprobe spacecraft. Experimental results that demonstrated the utility of the approach are presented, as well as some new insights into the behavior of incremental evolution which may explain its success.

1 Introduction

Evolutionary optimization algorithms such as genetic algorithms [4] are an approach to optimization that uses biologically-inspired selection, recombination and mutation operators to evolve a population of candidate solutions to a problem, in a process analogous to biological evolution. Much of the work on evolutionary optimization has focused on investigation of the mechanics of the search *algorithm*, i.e., the mechanisms by which the space of solutions is explored.

A complementary approach is to focus on the searchability of the solution space that is explored (that is, by the fitness function over the space of possible solutions). *Incremental evolution* [5, 3] is an extension to evolutionary optimization which decreases the computational effort of evolving the solution to a difficult problem by first evolving solutions to "easier" problems.

The intuition behind this approach is attractive:

- It is often easier to learn difficult tasks after simpler tasks have been learned; and
- It may therefore be advantageous to use an "easier" fitness function when evolving solutions to complex problems.

This intuition is consistent with the phenomenon of scaffolding, which has been studied in psychology [12].

Previous approaches that are related to incremental evolution include work in optimization in a dynamic environment ([11, 2, 10]), multi-phasic fitness environments ([1]), coevolution ([7]), and methods for test case selection for evolutionary algorithms [9, 13]. See [3] for a full discussion on the relationship between incremental evolution and these other approaches. Previous works that have discussed the application of incremental evolution techniques are [6, 5, 3].

This paper presents the application of incremental evolution to a real-world, spacecraft design optimization problem – the physical design optimization of a Mars soil penetrator microprobe. Based on the experimental results in this domain, new insights into the incremental evolution mechanism are presented. The rest of the paper is organized as follows: In Section 2, we review the incremental evolution technique. In Section 3, we describe the Mars microprobe design problem. Section 4 presents experimental results in the Mars microprobe domain, demonstrating that incremental evolution can yield significant performance improvements over conventional optimization, and an analysis of the results to better understand why incremental evolution is successful. Section 5 concludes the paper with a discussion and directions for future work.

2 Incremental Evolution

The essential idea of incremental evolution is to *scale* the evaluation function (i.e., the "fitness function" against which, say, a candidate solution is evolved) over time, with the aim of minimizing the overall time spent evolving a controller that achieves the prescribed task. Suppose that our goal is to generate, within a prescribed time limit T, a solution to optimize some evaluation function G. The problem of incremental evolution is to derive a set of intermediate evalua-

```
initialize($Pop$)
$F := G_0$
evaluate($Pop, F$)
for $evalfunc :=$ 0 to $k-1$
  $F := G_{evalfunc}$
  for $gen :=$ 0 to $t_{evalfunc}$
    $Pop :=$ mutate(recombine($Pop$))
    evaluate($Pop, F$)
  end for
end for
```

Figure 1: Algorithm schema for an incremental evolutionary algorithm. Pop is a population of candidate solutions; F is an evaluation function.

tion functions $\mathcal{G} = (G_0, G_1, \ldots, G_{k-1} = G)$ and a schedule $\mathcal{S} = (t_0, t_1, \ldots, t_{k-1})$, such that $t_0 + t_1 + \ldots + t_{k-1} = T$. The population of candidate solutions is sequentially evolved using evaluation function G_k for time t_k, beginning with G_0 for time t_0.

Let $\tau(\mathcal{G}, \mathcal{S}, Q)$ be the total processing effort (e.g., CPU time) required to evolve a solution of quality Q for the task G, given the sequence of tasks $\mathcal{G}$ and the schedule $\mathcal{S}$. Given any final evaluation function G and a desired solution quality Q, we wish to be able to choose $(\mathcal{G}, \mathcal{S})$ so that $\tau(\mathcal{G}, \mathcal{S}, Q)$ is minimized. This is a non-trivial, meta-level optimization, and a methodology for computing optimal $(\mathcal{G}, \mathcal{S})$ sequences for arbitrary G is unlikely. Indeed, certain choices of $(\mathcal{G}, \mathcal{S})$ may result in a performance degradation when compared with the trivial schedule that uses $\mathcal{G}' = (G)$ and $\mathcal{S}' = (t_0 = T)$, that is to say, $\tau(\mathcal{G}, \mathcal{S}, Q) > \tau(\mathcal{G}', \mathcal{S}', Q)$.

In this paper, we restrict our attention to *two-stage incremental evolution*, where there is only one intermediate task (i.e., evaluation function) and only one transition between evaluation functions. In other words, we use $k = 1$, $\mathcal{G} = (G_0, G_1)$, and $\mathcal{S} = (t_0, t_1 = (T - t_0))$; it is understood that G_1 is the final, or "target", evaluation function G.

Figure 1 shows a simplified, general schema for the k-stage incremental evolution, where *initialize, evaluate, recombine, mutate* are domain-specific operators applied to the population. For example, for two-stage incremental evolution, there is a transition between the two fitness functions G_0 and G_1 at generation t_0. *evaluate* computes the fitness values of the members of a population $P(t)$ with respect to fitness function F (which, in the case of two-stage incremental evolution, is either G_0 or G_1, depending on t).

3 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 ballistically enter the Martian atmosphere and passively orient themselves to meet peak heating and impact requirements. Upon impacting the Martian surface, the probes will punch through the entry aeroshell and separate into a fore- and aftbody system. 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 micropenetrator networks: ultra low temperature batteries, power microelectronics, and advanced microcontroller, a microtelecommunications system and a science payload package (a microlaser system for detecting subsurface water).

We applied incremental evolution to optimize the physical design parameters for the Mars microprobe. The microprobe optimization domain in its entirety is very complex, involving a three-stage simulation: *stage 1*- separation analysis (i.e., separation from the, Mars Surveyor), *stage 2*- aerodynamical simulation, and *stage 3*- soil impact and penetration. The complete design model for the penetrator is currently under development. Below, we describe the optimization of the current model, which implements the stage 3 (impact/penetration) problem.

Given a number of 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, where the objective is to maximize the ratio of the depth of penetration to the length of the penetrator. A fitness value of zero indicates a complete failure on behalf of the probe (e.g., a design that would bounce off the target surface). A negative fitness value indicates a design that is not physically realizable – note that this is not possible to determine a priori without consulting the simulation.

4 Experimental Results

To evolve solutions for the Mars microprobe design optimization problem, we used a standard, generational genetic algorithm using one-point crossover and and a bit-flipping mutation operator [4]. The population size was 50, and each run was 100 generations. The crossover rate was 0.6, and the mutation rate was 0.01 per bit. Each parameter was encoded as a 32-bit bit string.

4.1 Generating New Evaluation Functions for Incremental Evolution

In order to generate new evaluation functions for incremental evolution (i.e., G_0), we consider the initial condition variables which specifies the hardness of the target surface (measured by a *soil number* – the higher the soil number,the softer the surface). Intuitively, the softer the target surface, the easier it is to penetrate the surface.

Thus, for our two-stage incremental evolution experiments, we generate G_0 as follows: G_0 is identical to the "true evaluation function" (G_1), except that in the soil penetration simulation, target soil number is set differently than for G_1 (For G_1, soil number = 7). soil number was varied between 3 (very hard) and 14 (soft).

In all of the experiments below, we set $t_0 = 30$. That is, the transition between G_0 and G_1 occurred after the 30th generation.[1].

4.2 Incremental Evolution Results

We say that (G_0, t_0) successfully *primes* for G_1 if $\tau((G_0, G_1), (t_0, t_1), Q) < \tau((G_1), (t_0 + t_1), Q)$, i.e., the incremental evolution reduces the time required to reach the prescribed solution quality Q.

Figure 2 shows the performance of incremental evolution for various G_0, where the soil number used to determine G_0 was varied between 3 and 14 (hereafter, we say "soil=x" to denote the choice of G_0 for which the soil number is x.) The control experiment is soil=7 (i.e., $G_0 = G_1$), which corresponds to the standard genetic algorithm. Each curve shows the mean of 30 independent runs of incremental evolution. In general, the higher soil numbers for G_0 were correlated with better performance, that is, G_0 with higher soil numbers successfully primes for soil=7. Soil=14 significantly outperformed the control case (soil=7), while (soil=3) performed significantly worse than soil=7.

4.3 Analysis

In order to gain some insights into the results of the previous section, we analyze the cost surfaces used for G_1. Figure 3-5 show sampled portions of the G_1 cost surfaces used in the experiments above. The sampling resolution was one point every 0.02 inches for outside diameter, and 0.04 feet of total penetrator length.

Several observations can be made about the structure of the cost surfaces. First, in general, the cost surfaces are quite "rugged", in that points with positive fitness are interspersed with points of zero and negative fitness.[2] This is a significant factor in determining the relative difficulty of a cost surface for an evolutionary algorithm. Second, there is an apparent correlation between the soil number and the ruggedness of the

[1]We chose this transition schedule because it was shown empirically in [3] that two-stage incremental evolution yielded the best performance when the transition between G_0 and G_1 occurred at an early stage of the optimization

[2]We emphasize that the figures show sampled points on the cost surface, and are not meshes.

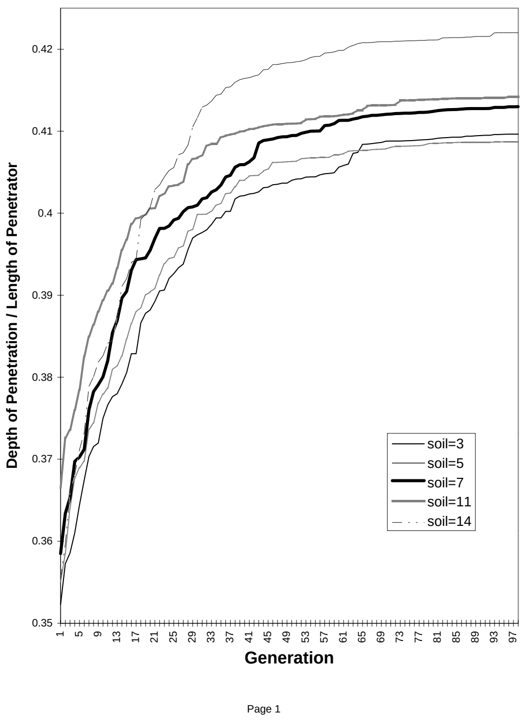

Figure 2: Performance of incremental evolution using priming (G_0) functions where the soil number was varied between 3 (hard soil) and 14 (soft soil). The control case (i.e., $G_0 = G_1$ is soil=7). Each curve shows the mean of 30 independent runs.

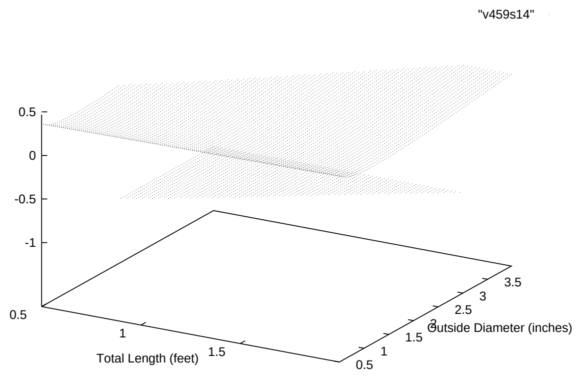

Figure 3: Sampled cost surface of G_0, where soil = 14 (soft soil). The z-axis represents the fitness value.

surface. The cost surfaces for G_0 with higher soil numbers

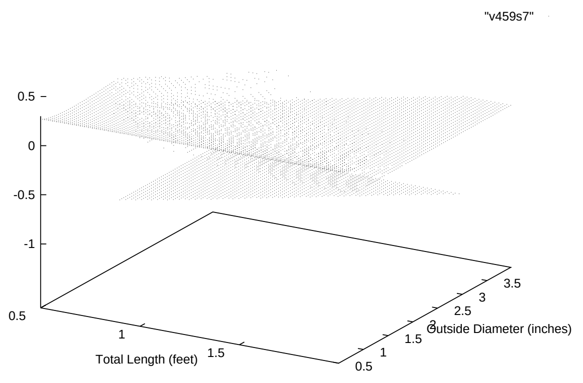

Figure 4: Sampled cost surface of G_0, where soil = 7 (moderately hard soil). The z-axis represents the fitness value.

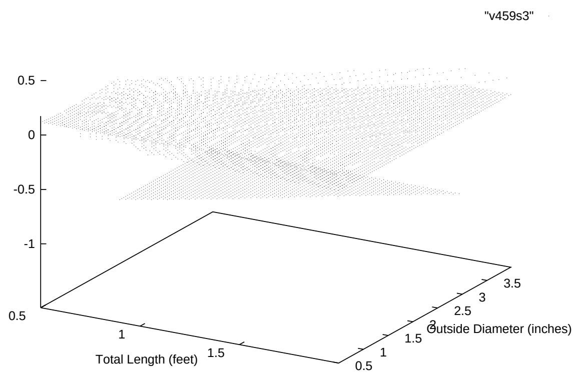

Figure 5: Sampled cost surface of G_0, where soil = 3 (very hard soil). The z-axis represents the fitness value.

are smoother than the cost surfaces for G_0 with lower soil numbers.

For example, compare the cost surface for soil=14 (Figure 3) versus the cost surface for soil=3 (Figure 5). For soil=14, the cost surface consists of two "planes": The top plane (where the fitness values are above zero) represents the set of physically realizable designs, all of which successfully penetrate the target surface, while the lower plane (where the fitness values are below zero) represents the set of designs which are physically unrealizable. In constrast, the cost surface for soil=14 consists of three planes. As with soil=3, there is the set of designs whose fitnesses are positive, and a set of designs whose fitnesses are negative.[3] In addition, some designs yield a fitness of zero (mission failure), resulting in a third plane at fitness=0. This results in an obviously more irregular ("bumpier") cost surface than for soil=3.

Finally, it is clear that the overall structure of the surfaces are quite similar, e.g., the global maxima of the samples are around (length=0.5 feet, diameter=0.5 inches) in all cases.

This suggests a possible explanation for the behavior of incremental evolution in this domain. First, choosing a G_0 that is smoother than G_1 in effect "smooths" the search space for the evolutionary algorithm, providing an easier cost surface. Second, if the overall structures of G_0 and G_1 are similar, then finding good solutions in the space of G_0 correspond to finding good solutions in the space of G_1. Thus, if G_0 is chosen appropriately, incremental evolution is like searching a smoothed approximation of G_1 that is easier to search than G_1; because the evolutionary algorithm is less likely to get stuck in local minima in the less bumpy surface of G_0, progress towards a global near-optimum is faster.

[3] noted the need for a theory of the relationship between "problem difficulty" and incremental evolution success. We now have the beginnings of such a theory, based on cost surface structure. Although the first studies of incremental evolution [3, 6, 5] were clearly inspired by the intuitive notion of "learning easier tasks before more difficult tasks", e.g., it is "easier" to penetrate the target surface if the target is softer, the results of this paper and [3] strongly suggest that such naive notions of "easy" and "hard" problems are not sufficient to predict the success of incremental evolution. To develop a normative theory of incremental evolution, attention should be focused not on easy/hard "problems" (in the intuitive sense), but on the analysis of easy/hard cost surfaces for a particular search algorithm.

5 Discussion and Future Work

The major contributions of this paper has been to demonstrate the utility of incremental evolution for real-world optimization problems, and to offer some new insights that could explain how the mechanism operates. Although [3] demonstrated statistically significant improvements (over standard evolution) when using incremental evolution in two artificial optimization domains (the Tracker [8] foraging domain and a pursuit-evasion game), the results were somewhat mixed, in that it was not clear under what circumstances the approach would be successful.

The results in this paper provide support that incremental evolution is a viable approach in practice for real-world optimization problems – a genetic algorithm with incremental evolution was able to find higher-quality solutions in less time

[3]Note that the region of physically unrealizable designs is identical, since whether a design is realizable or not is independent of environmental conditions such as soil hardness.

than a standard genetic algorithm on the Mars microprobe design problem.

In addition, this paper has yielded some interesting insight into how the incremental evolution mechanism operates. Since the domain is a problem with only two decision variables (probe length and outer diameter), it was possible to visualize the cost surfaces of the priming functions (this was not possible in the higher-dimensional problems studied in [3]). Our analysis suggest two factors that are important in order for priming (improved performance) to be observed in the application of incremental evolution:

- The structure of the priming and target evaluation functions should be similar (with respect to the location of the near-optimal points); and
- The structure of the priming function cost surface should be "easier" to search than the target evaluation function.

In future work, we will test these conjectures by investigating the behavior of incremental evolution using some synthetic cost functions for which we can easily control the above characteristics of the priming functions.

Currently, incremental evolution is applied by manually choosing the priming functions and the schedule (i.e., G_0 and t_0), and observing the results. One could automate this procedure by iteratively choosing values of G_0 and t_0. Although this would be a time-consuming procedure, it can be worthwhile if higher-quality solutions can be found using incremental evolution.[4] Since we now have a qualitative theory (based on the priming function cost surface structure) that predicts the success incremental evolution, we are currently investigating methods for exploiting the theory to efficiently automate the incremental evolution process to some extent. For example, given a G_0, it may be possible to sparsely sample its cost surface and estimate the degree to which the two conditions described above are satisfied. These estimates can then be used to prioritize the order in which priming functions are used for G_0.

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. Bob Glaser provided the simulation software for the Mars microprobe model. Thanks to Andre Stechert and the anonymous reviewers for helpful comments and suggestions on this paper.

[4]Note that in Figure 2, incremental evolution is discovering better solutions than those found by the conventional method, by the time the algorithms apparently converge.

References

[1] D. Andre. Evolution of mapmaking: Learning, planning and memory using genetic programming. In *Proc. IEEE International Conf. on Evolutionary Computation (ICEC)*, pages 250–255, 1994.

[2] H.G. Cobb and J.J. Grefenstette. Genetic algorithms for tracking changing environments. In *Proc. Fifth International Conference on Genetic Algorithms*, pages 523–530, 1993.

[3] 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.

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

[5] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: Artificial evolution, real vision. In *From Animals to Animats 3: Proceedings of the Third International Conference on Adaptive Behavior*, pages 392–401, 1994.

[6] I. Harvey, P. Husbands, and D.C. Cliff. Issues in evolutionary robotics. In *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 364–374, 1992.

[7] D. Hillis. Co-evolving parasites improve simulated evolution. *Physica D*, 42:228–234, 1990.

[8] D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wang. Evolution as a theme in artificial life: The genesys/tracker system. In C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 549–577. Addison-Wesley, 1992.

[9] J. Koza. *Genetic Programming: On the Programming of Computers By the Means of Natural Selection.* MIT Press, 1992.

[10] M.L. Littman and D.H. Ackley. Adaptation in constant utility non-stationary environments. In *Proc. Fourth International Conference on Genetic Algorithms*, pages 136–142, 1991.

[11] C.L. Ramsey and J.J. Grefenstette. Case-based initialization of genetic algorithms. In *Proc. Fifth International Conference on Genetic Algorithms*, pages 84–91, 1993.

[12] J. Rutkowska. Emergent functionality in human infants. In *From Animals to Animats3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 179–188. MIT Press, 1994.

[13] A. C. Schultz. Adapting the evaluation space to improve global learning. In *Proc. Fourth International Conference on Genetic Algorithms*, pages 158–164, 1991.