In Proceedings of IEEE Aerospace Conference, Snowmass, CO, 1997

Automating the Process of Optimization in Spacecraft Design

Alex S. Fukunaga, Steve Chien, Darren Mutz,
Robert L. Sherwood, Andre D. Stechert
Jet Propulsion Laboratory, MS 525-3660
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
(818)306-6157
alex.fukunaga@jpl.nasa.gov, steve.chien@jpl.nasa.gov, darren.mutz@jpl.nasa.gov,
robert.sherwood@jpl.nasa.gov, andre.stechert@jpl.nasa.gov

Abstract – Spacecraft design optimization is a difficult problem, due to the complexity of optimization cost surfaces and the human expertise in optimization that is necessary in order to achieve good results. In this paper, we propose the use of a set of generic, metaheuristic optimization algorithms (e.g., genetic algorithms, simulated annealing), which is configured for a particular optimization problem by an adaptive problem solver based on artificial intelligence and machine learning techniques. We describe work in progress on OASIS, a system for adaptive problem solving based on these principles.

TABLE OF CONTENTS

1. INTRODUCTION

Many aspects of spacecraft design can be viewed as instances of *constrained optimization problems*. Given a set of decision variables X and a set of constraints C on X, the constrained optimization is the problem of assigning values to X to minimize or maximize an objective function $F(X)$, subject to the constraints C.

Spacecraft design optimization is difficult using current optimization methods because:

- Current methods require a significant amount of manual customization by the users in order to be successful, and
- Current methods are not well suited for mixed discrete/continuous, non-smooth, and possibly probabilistic cost surfaces that can arise in many design optimization problems.

We are currently developing the Optimization Assistant (OASIS), a tool for automated spacecraft design optimization that addresses these two issues. The goal of OASIS is to facilitate rapid "what-if" analysis of spacecraft design by developing a widely applicable, spacecraft design optimization system that maximizes the automation of the optimization process and minimizes the amount of customization required by the user.

OASIS consists of an integrated suite of global optimization algorithms that are appropriate for non-smooth, possibly probabilistic, mixed discrete/continuous cost surfaces, and an intelligent agent that decides how to apply these

algorithms to a particular problem. Given a particular spacecraft design optimization problem, OASIS performs a meta-level optimization in order to:

- Select an appropriate optimization technique to apply to the problem, and
- Automatically adapt (customize) the technique to fit the problem.

The rest of this paper is organized as follows. Section 2 describes the application of metaheuristic algorithms to optimization, and its problems. In Section 3, we define the framework of adaptive problem solving that we adopt for OASIS and describe related work in the area. Section 4 presents an overview of the OASIS system architecture and describes our approach to solving the adaptive solving problem task. In Section 5, we describe two spacecraft design optimization problems which are currently being used as testbed applications for OASIS: the NASA New Millennium DS-2 Mars Microprobe and the Neptune Orbiter spacecraft.

2. Optimization Using Metaheuristics

Although optimization is a mature field that has been studied extensively by researchers, there are a number of open, fundamental problems in the practical application of optimization techniques.

First, the problem of global optimization on difficult cost surfaces is poorly understood. The optimization of smooth, convex cost functions is well understood, and efficient algorithms for optimization on these surfaces have been developed. However, these traditional approaches often perform poorly on cost surfaces with many local optima, since they tend to get stuck on local optima. Unfortunately, many real-world optimization problems have such a "rugged" cost surface and are thus difficult problems for traditional approaches to optimization.

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 possible to directly analyze the cost surface by analytic means in order to guide an optimization algorithm. For example, *F(X)* can be 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 legacy FORTRAN code about which very little is known to the optimizer except for its I/O specifications). Black-box optimization problems are therefore challenging because currently known algorithms for black-box optimization are essentially "blind" search algorithms—instead of being guided by direct analysis of the cost surface, they must sample the cost surface in order to indirectly obtain useful information about the cost surface.

Recently, there has been much research activity in so-called *metaheuristic* algorithms such as simulated annealing [15], tabu search [7,8] and genetic algorithms [9] for global optimization. These are loosely defined, "general-purpose" heuristics for optimization that proceed by iteratively sampling a cost surface, and they implement various mechanisms for escaping local optima. Although these algorithms have been shown to be successful on numerous applications with difficult cost surfaces, the behavior of these algorithms is still poorly understood. Successful application of these metaheuristics[1] to a particular problem requires:

- Selection of the most appropriate metaheuristic for the problem, and
- Intelligent configuration of the metaheuristic by selecting appropriate values for

[1] In the rest of the paper, we use the terms *metaheuristic* and *metaheuristic algorithm* interchangeably.

various control parameters (e.g., temperature cooling schedule for simulated annealing).

Currently, successful applications of metaheuristics are often the result of an iterative cycle in which a researcher or practitioner selects and adjusts a number of different metaheuristic/control parameter combinations on a problem, observes the results, and repeats this process until satisfactory results are obtained. This process of selecting and configuring a metaheuristic to obtain good results on a given problem is usually time-consuming, and requires a significant amount of optimization expertise (which is often very costly to obtain). As a result, in many cases, the cost of successfully applying metaheuristic techniques on black-box problems can be prohibitively expensive.

One might wonder whether there is some super-metaheuristic and a perfect configuration of this super-metaheuristic, which outperforms all others for all problems of interest, or whether it is at least possible to characterize the performance of metaheuristic configurations in general. The current conventional wisdom in the optimization research community is that this possibility is extremely unlikely (although it not likely that this can ever be formally proved, due to the empirical nature of the question).[2] This is supported by related recent theoretical work such as [24], which shows that over all possible cost surfaces, the expected performances of all optimization algorithms are exactly equal. Although it is possible that "all problems of interest" (in our context, all nontrivial spacecraft design optimization problems) reflect a particular subset of all possible cost surfaces for which some metaheuristic configuration's performance dominates that of all others, we strongly believe that this is not the case. *Thus, our assumption throughout this paper is that to obtain the best performance for a particular problem instance, it is necessary to select a metaheuristic and configure it so that it matches the structure of the cost surface of the instance*.

3. Adaptive Problem Solving

A natural approach to alleviating this problem of selecting and configuring a metaheuristic for particular applications is to automate the process. This is an instance of the more generic, *adaptive problem solving* task, which has been studied by the artificial intelligence community, where the task is to automatically configure a problem solving system (such as an optimization system). In this section, we give the standard definition of the adaptive problem solving task, and review previous approaches in the literature[3]. We then discuss a generalization of adaptive problem solving, which is the framework we will adopt for the metaheuristic application problem in spacecraft design optimization.

Before discussing approaches to adaptive problem solving, we formally state the standard definition of the task (as proposed by [10,11,12,17,23]. Adaptive problem solving requires a *configurable* problem solver, meaning the problem solver possesses control

[2] Nevertheless, it is not difficult to find in the metaheuristic literature empirical studies that claim that one metaheuristic or one configuration is better than another (e.g., [25] boldly claims that "the objective of this paper is...to study the general tendencies of various algorithms," and proceeds by comparing the performance of several metaheuristics on a scheduling problem. They conclude: "If obtaining solutions of higher quality is important, use Simulated Annealing or Greedy Local Search. Detailed parameter tuning is not important for Simulated Annealing and Greedy Local Search provided that sufficient amount of computational time is available."

[3] The formal statement of the traditional adaptive problem solving forumulation and the review of previous work is based on the treatment in [10].

decisions that may be resolved in alternative ways. Given a configurable problem solver, *PS*, with several control points, $CP_1 ... CP_n$ (where each control point CP_i corresponds to a particular control decision), and a set of values for each control point, $\{M_{i,1} ... M_{i,k}\}$[4], a *control strategy* is an assignment of values to control points that defines the overall behavior of the problem solver. Let PS_{STRAT} be the problem solver operating under a particular control strategy.

The quality of a problem solving strategy is defined in terms of the decision-theoretic notion of expected utility. Let $U(PS_{STRAT},d)$, be a real valued utility function that is a measure of the goodness of the behavior of the problem solver on a specific problem d.[5] Then, the expected utility can be defined formally over a distribution of problems D:

$$E_D[U(PS_{STRAT})] = \sum_{d \in D} U(PS_{STRAT}, d) \times pr(d)$$

The goal of this standard formulation of adaptive problem solving can be expressed as: given a problem distribution D, find some control strategy in the space of possible strategies that maximizes the expected utility of the problem solver. For example, for the problem of configuring a metaheuristic, say, a genetic algorithm, in a design optimization system, the control points include: the population size, the crossover rate, and the mutation rate, etc. Utility might be defined as the quality of the design generated by the optimizer.

Note that a number of approaches to adapting control points such as the population size of a GA [21], have been proposed in the literature. In our framework, we consider such strategies to be values of control points (e.g., a particular implementation of an adaptive temperature schedule is one of the possible values for the temperature schedule control point for the simulated annealing metaheuristic).

Several approaches to adaptive problem solving have been discussed in the literature. The first, a *syntactic approach*, is to preprocess a problem-solving domain into a more efficient form, based solely on the domain's syntactic structure. For example, Etzioni's STATIC system analyzes a portion of a planning domain's deductive closure to conjecture a set of search control heuristics [3]. Dechter and Pearl describe a class of constraint satisfaction techniques that preprocess a general class of problems into a more efficient form [2]. More recent work has focused on recognizing those structural properties that influence the effectiveness of different heuristic methods [4,14,22]. The goal of this approach is to provide a problem solver with what is essentially a big lookup table, specifying which heuristic strategy to use based on some easily recognizable syntactic features of a domain. While this latter approach seems promising, work in this area is still preliminary and has focused primarily on artificial applications. The disadvantage of purely syntactic techniques is that they ignore a potentially important source of information, the distribution of problems. Furthermore, current syntactic approaches to this problem are specific to a particular, often unarticulated, utility function (usually problem-solving cost). For example, allowing the utility function to be a user-specified parameter would require a significant and problematic extension of these methods.

The second approach to adaptive problem solving, the *generative approach*, is to generate custom-made heuristics in response to careful, automatic, analysis of past problem-solving attempts. Generative approaches con-

[4] Note that a method may consist of smaller elements so that a method may be a set of control rules or a combination of heuristics.

[5] We assume that the problem solver runs for a finite amount of time and eventually terminates.

sider not only the structure of the domain, but also structures that arise from the problem solver interacting with specific problems from the domain. This approach is exemplified by SOAR [16] and PRODIGY/EBL [18]. These techniques analyze past problem-solving traces and conjecture heuristic control rules in response to particular problem solving inefficiencies. Such approaches can effectively exploit the idiosyncratic structure of a domain through this careful analysis. The limitation of such approaches is that they have typically focused on generating heuristics in response to particular problems and have not addressed the issue of adapting to a distribution of problems well[6]. Furthermore, as with the syntactic approaches, thus far they have been directed towards a specific utility function.

The third approach is the statistical approach. These techniques explicitly reason about performance of different heuristic strategies across the distribution of problems. These are generally statistical generate-and-test approaches that estimated the average performance of different heuristics from a random set of training examples and explore an explicit space of heuristics with greedy search techniques. Examples of such systems are COMPOSER [11], PALO [12], and the statistical component of MULTI-TAC [19]. Similar approaches have also been investigated in the operations research community [26]. These techniques are easy to use, apply to a variety of domains and utility functions, and can provide strong statistical guarantees about their performance. They are limited, however, as they are computationally expensive, require many training examples to identify a strategy, and face problems with local optima in the space of control strategies. Furthermore, they typically leave it to the user to conjecture the space of heuristic methods (see [19] for a notable exception).

A Generalization of Adaptive Problem Solving

The standard formulation of adaptive problem solving described above is applicable when we want to generate a problem solver that will perform well for a particular problem distribution. There are some problems with this formulation, however, that make it inappropriate for our domain of metaheuristic application for spacecraft design optimization.

First, although the strategy found by an adaptive problem solver may have good expected performance over some distribution of problem instances, there is no guarantee that the problem solver performs well for any particular instance. In the domain of design optimization, the objective is often to generate the best possible solution for a specific problem instance, so there is a significant incompatibility in the objective of the problem formulation.

Second, the standard adaptive problem solving formulation implicitly assumes that only one specific configuration of the problem solver will be applied to a particular problem instance. If the objective is to generate the best possible solution for a problem, then it may be worthwhile to try a number of different problem solver configurations on the problem instance. In design optimization, it is often worthwhile to use massive amounts of computing resources[7] in order to make significant improvements in the quality of the design, which could lead to benefits that far outweigh

[6] While generative approaches can be trained on a problem distribution, learning typically occurs only within the context of a single problem. These systems will often learn knowledge which is helpful in a particular problem but decreases utility overall, necessitating the use of utility analysis techniques.

[7] CPU cycles are often quite cheap and readily available, given the amount of computation available on idle workstations in many engineering organizations.

the computational resources used to generate the improvement.

Finally, the problem solver configuration found by the standard adaptive problem solving formulation is useful when we are given a problem instance that is "typical" of the instances in the distribution for which the problem solver was configured. This, however, may be of limited utility if the problem solver is faced with an instance which is significantly different from previously seen instances. This is a problem in our domain, since we are designing a generic design optimization tool for which the distribution is virtually unrestricted. One could argue that if an instance is sufficiently different from the distribution for which the configuration was optimized, then this forms the basis for a new distribution on which to run the adaptive problem solver (where initially, the distribution consists of this single, new instance). Of course, if we allow for the possibility of maintaining multiple problem solver configurations, one of which should be selected depending on the distribution to which a particular instance belongs, new subproblems arise that must be solved, including:

- Given a new problem instance, decide which distribution it belongs to.
- Deciding when/whether to "split" an existing problem distribution into two or more distributions when additional problems are added to the distribution.

For our problem of metaheuristic application for design optimization, what is needed then is a task formulation that maximizes the performance for each particular problem instance, and does not rely on initial assumptions about the problem distribution from which the instance is drawn. Our formulation for adaptive problem solving is therefore the following:

Definition: (Adaptive Problem Solving)—Let d be a problem instance. Let PS_{STRAT} be the problem solver operating under a particular control strategy. Let $U(PS_{STRATt},d)$, be a real valued utility function that is a measure of the goodness of the behavior of the problem solver on d. The task of adaptive problem solving is to find a control strategy for the problem solver that maximizes $U(PS_{STRAT},d)$. Given a set of problem instances $D=d0, d1,...dn$, the task of adaptive problem solving is to find a set of control strategies $Strat0, Strat1,... StratN$, that maximizes:

$$\sum_{d \in D} U(PS_{STRAT_d}, d)$$

If we were to treat the set of instances in the definition above as being samples drawn from a distribution, this formulation of adaptive problem can be seen as a generalization of the standard formulation, without the restriction that

$$Strat0 = Strat1 = StratN.$$

The approaches for the standard formulation can now be reevaluated with respect to the new formulation. In general, if we know of a configuration $Conf_D$ that maximizes expected utility over a distribution D of problems to which a particular instance d belongs, then one would, by definition, expect (in the probabilistic sense) that configuration to perform well on the instance. Suppose that our approach to solving the new adaptive problem solving formulation is to search the space of configurations to find a near-optimal control strategy. Then, a useful heuristic would be to try $Conf_D$ first. Thus, we can treat solutions to the standard formulation as heuristic solutions for our formulation of adaptive problem solving.

4. OASIS ARCHITECTURE

OASIS (Optimization Assistant) is an integrated software architecture for spacecraft design optimization that supports adaptive problem solving. The three major components of OASIS are:

- A spacecraft design model,
- A suite of configurable metaheuristics, and
- An adaptive problem solver.

We describe each of these in the following discussion.

Spacecraft Design Model

The 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 the component of OASIS that is the most domain-specific, and is provided by the end users, i.e., spacecraft designers. In order for an optimization system such as OASIS 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 to be imposed, the optimization system will not be used by spacecraft designers.

The Multidisciplinary Integrated Design Assistant for Spacecraft (MIDAS) [6] is a graphical design environment that allows a user to integrate a system of possibly distributed design model components together using a *methogram*, a graphical diagram representing 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, or 3) an embedded methogram (i.e., this allows methograms to have a hierarchical structure). 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. MIDAS is implemented as a CORBA object, and supports a wide variety of methods that can be used by external client systems (e.g., a GUI) to manipulate the methograms.

This last feature of MIDAS (i.e., the CORBA interface that allows client systems to freely manipulate methograms) is particularly useful for the purposes of designing a black-box optimization system, since it essentially provides the optimization system with a uniform interface for any design model encapsulated in MIDAS. Therefore, our solution to the problem of supporting a wide range of design models is to support an interface to MIDAS. That is, OASIS is designed to be an optimization system that can be used to optimize any MIDAS model.

Thus, the design model, which constitutes the user input to the OASIS sytem, is composed of the following:

- A MIDAS methogram that encapsulates the design model,
- A list of decision variables, as well as ranges of their possible values (may be continuous or discrete), and

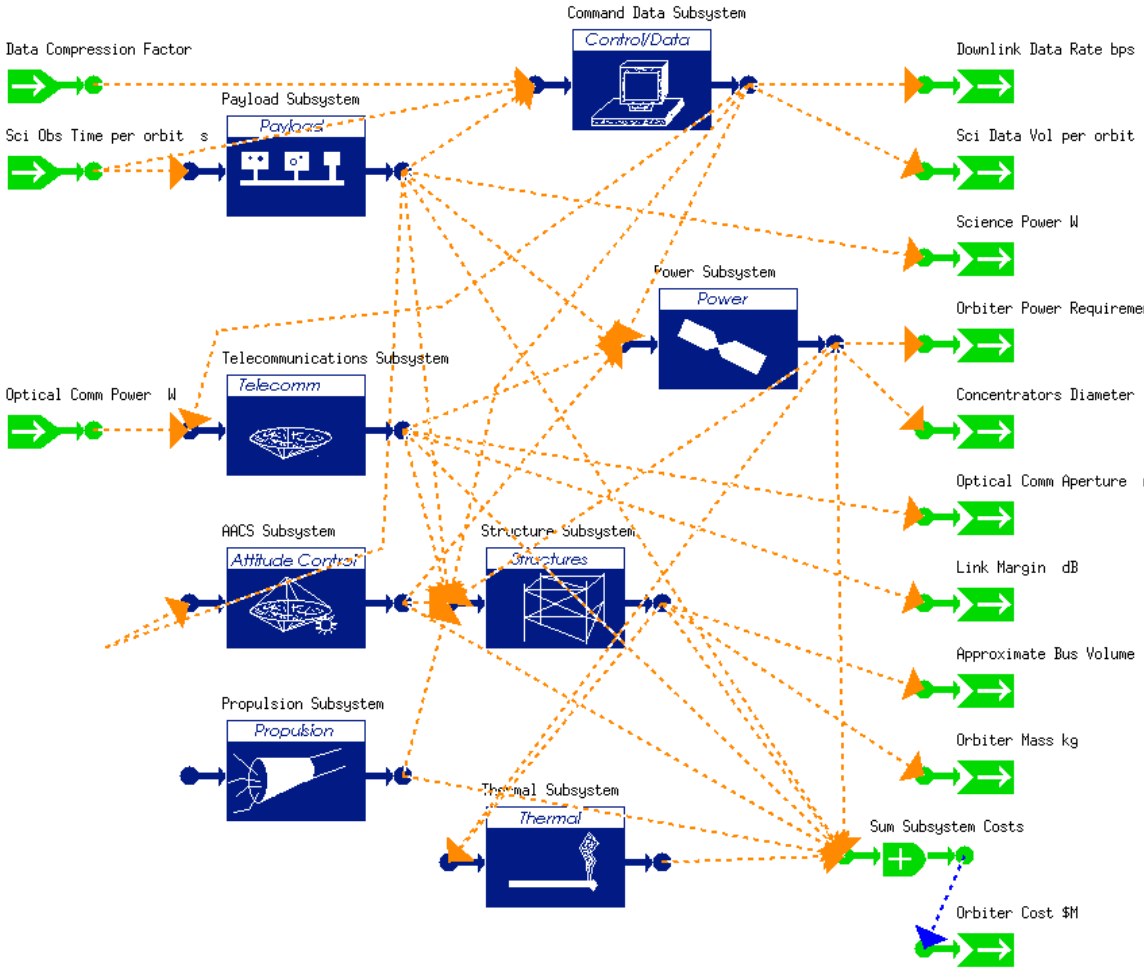

Figure 1—**Screen shot of a MIDAS methogram (part of the Neptune Orbiter model)**

- An output from a methogram node that corresponds the user's objective function value.[8]

Figure 1 shows part of a MIDAS methogram for the Neptune Orbiter model (see Section 5).

Metaheuristic Suite

OASIS includes a set of *configurable metaheuristics*, which are generic implementations of metaheuristics that provide an interface for dynamic reconfiguration of their control points at runtime. Currently, this consists of a reconfigurable genetic algorithm and a reconfigurable simulated annealing/local search. These are briefly described below[9]:

Genetic Algorithm—A genetic algorithm works as follows: a population of sample points from the cost surface is generated. In a process analogous to biological evolution, this

[8] The objective function could either be obtained directly from one of the existing outputs in the methogram, or it could be computed by adding a new node that computes, e.g., a weighted linear combination of some set of output nodes.

[9] The following is a simplified account - there is much overlap between metaheuristics, and their boundaries are unclear (it is often possible to consider one metaheuristic as a specialization/generalization of another). For example, it is possible to think of some instances of local search as a special case of genetic algorithm with a population of 1. For clarity, we present metaheuristics using their "canonical" descriptions.

population is evolved by repeatedly selecting (based on relative optimality) members of the population for reproduction, and recombining/mutating to generate a new population. The control points of the genetic algorithm include: the population size, the methods used for selection, crossover, and mutation (methods include the algorithm, as well as their control parameters, such as their frequency of application).

Simulated Annealing/Local Search—Local search proceeds by generating an initial point on the cost surface and repeatedly applying neighborhood moves (such as random perturbations, greedy moves, etc.) to move to (on average) increasingly optimal points on the cost surface. Simulated annealing is a generalization of local search inspired by a physical metaphor to the process of annealing, in which moves to less optimal points are taken probabilistically, in accordance with a temperature schedule. During the early part of the annealing process, the temperature is high, and moves to less optimal points are taken more frequently; as the temperature is decreased, the probability of rejecting moves to poorer points on the cost surface increases. The control points of local search/simulated annealing include methods for temperature schedule and the neighborhood move generator.

Adaptive Problem Solver

Given a spacecraft design optimization problem instance in the form of a design model, the adaptive problem solver component of OASIS selects and configures a metaheuristics from its suite in order to maximize some utility measure (usually, this is the quality of the design found by OASIS). In this section, we motivate our approach to adaptive problem solving and describe the architecture of the OASIS adaptive problem solver.

Our approach to adaptive problem solving is to view it as a meta-level heuristic search through the space of possible problem solver configurations, where candidate configurations are evaluated with respect to a utility measure, and the goal is find a configuration that maximizes this utility measure. In principle, it is possible to do a brute-force search through the space of possible problem solver configurations. This method is clearly intractable in general, since the number of configurations is exponential in the number of control points. Consider a problem solver with c control points, each with v values; there are c^v problem solver configurations. Suppose we are given a problem instance for which the black-box simulation runs to evaluate a single candidate design takes an average of t seconds. If each run of the problem solver on this instance requires n candidate design evaluations on average, then the expected time required to search this space using an unguided, brute force search is $O(ntc^v)$.

Given the enormous computational expense of searching through the space of problem solver configurations, one might wonder whether the search should/could be avoided altogether. To avoid search completely, there are two alternatives. The first is to find a super-metaheuristic that outperforms all others for all problem instances (and thereby avoid adaptive problem solving altogether). As discussed in Section 2, we reject this solution as infeasible. The second alternative is a syntactic, "lookup-table" approach: classify the problem instance as a member of some class of problems, then apply the metaheuristic configuration that is known to work well for this class of problems. This method can work very well if we happen to have studied the class of problems to which the particular instance belongs, and we have available a good technique for classifying the instance as a member of the class. This approach, however, is of limited

utility if we encounter an instance of a class that we know nothing about, or if we cannot correctly classify the problem as one that belongs to a class for which we have a good metaheuristic configuration.[10] Thus, a purely syntactic approach does not suffice. *An adaptive problem solver needs to search the space of possible metaheuristic configurations—the challenge is to discover and apply enough heuristic knowledge to the task to make it more tractable.*

What types of heuristic knowledge are available to be either acquired from a human, or through knowledge discovery/machine learning techniques? We identify three general classes of knowledge which are applicable in this context:

- domain-dependent knowledge about the behavior of metaheuristics in a given domain,
- domain-independent, meta-knowledge about optimization, and
- domain-independent, but system-dependent structural knowledge.

Each of these types of knowledge are discussed below:

Domain-dependent knowledge—This includes knowledge about the structure of particular classes of problems, and the behavior of metaheuristics on particular problem instances or classes of instances. Examples of heuristics that can be derive from this type of knowledge include:

- If a problem instance is in the problem instance class *I*, then configuration *C* is promising;
- If a problem instance *i* is in the problem instance class *I*, then it is likely that its cost surface is of type *S*;
- If the behavior of a configuration *C* is poor, then the instance is likely to be in instance class *I*;
- If the cost surface of the instance belongs to class *S*, then the instance is likely to be in instance class *I*.

In addition, this class includes any knowledge that can be used to classify a problem instance as belonging to a particular class of problems, including pattern classification heuristics that can be used by a problem instance analysis module.

Domain-independent knowledge—This is a formalization of the knowledge that human optimization experts possess about optimization in general. It includes knowledge about the behavior of metaheuristics on particular classes of cost surfaces[11], and knowledge about the behavior of metaheuristics in general. Classes of cost surfaces can be defined by attributes such as the number of local minima, distance relationships between local minima, etc. Examples of heuristics that can be derived from this type of knowledge include:

- If a surface of class *S*, configuration *C* is promising;
- If the behavior of a configuration *C* is poor, then, configuration *C'* is promising;
- Given some features of the observed behavior of configuration *C*, it is likely that the cost surface is in class *S*.

[10] Indeed, the problems of defining useful notions of classes of problem instances, and classifying a problem instance as belonging to some particular class is a challenging pattern recognition problem in itself.

[11] Note the difference between *classes of problem instances*, and *classes of cost surfaces*. Cost surfaces are a more abstract, and classes of cost surfaces can encompass many classes of problems. For instance, the class of cost surfaces that are "bumpy or rugged" includes cost surfaces from a very large number of classes of problem instances.

This class of knowledge may not be as powerful as domain-dependent knowledge by itself, since it is more abstract in nature and more difficult to apply. For example, analysis of the cost surface is more computationally expensive than syntactic analysis of the instance, since it requires expensive runs of the black-box simulation to evaluate the cost surface. However, human optimization experts who may not be domain experts for a particular problem that they are given must often rely on their body of domain-independent knowledge, in combination to what domain-dependent knowledge they can obtain. The apparent success of these optimization experts indicates that this type of knowledge can be a very useful means of controlling search.

Domain-independent, system-dependent structural knowledge—It may be possible to exploit the syntactic structure of the problem representation in a particular adaptive problem solver. For instance, in OASIS, for a particular problem instance, it may be possible to analyze the structure of its dataflow graph in the MIDAS methogram to identify, e.g., decision variables that affect a relatively large number of other nodes in the graph. Hence, it may be more important to focus on nodes of high degree than a node than a node of low degree. At this time, it seems that useful knowledge of this type may be extremely difficult to acquire, although this is an area of research that seems particularly interesting from the design/human-computer interface perspectives, since it entails the understanding of how engineers structure simulations from a graph-theoretic point of view.

The various types of knowledge described above can be applied either directly or indirectly (through chains of inference) as variable/value ordering heuristics[12] in a search algorithm that searches the space of metaheuristic configurations.

We are currently investigating several alternative approaches to representing the knowledge in OASIS. The initial version of the OASIS Adaptive Problem Solver uses Bayesian networks [20] as its primary knowledge representation scheme. Bayesian networks were chosen primarily due to their clear, probabilistic semantics, the flexibility with which various kinds of inferences could be performed, and the ability to seamlessly integrate new knowledge and observations into the knowledge base either manually (i.e., entered by a human expert) or automatically (using machine learning techniques).

In the initial implementation, the Bayesian networks are manually constructed. A number of techniques have recently been developed for learning in Bayesian networks [13]; we are currently investigating the application of these techniques. In addition, we are investigating new approaches to machine learning that take advantage of the special structure of the adaptive problem solving domain. For example, unlike most other domains, in which all the training data for machine learning is given to a learning algorithm by an external source, the adaptive problem solving domain is interesting in that it is possible for the problem solving system to perform experiments and *generate* new data, using the exact same mechanisms that are used to evaluate metaheuristic configurations.

An important auxiliary component that is important for the adaptive problem solving process and applicable at various levels of the OASIS architecture is the hypothesis testing

[12] Variable ordering heuristics guide the choice of control points to change; value ordering heuristics guide the choice of control point values to try.

module. *Hypothesis testing* is a statistical problem in which the confidence in the choice of one belief over another given data is quantified. When, for example, a metaheuristic is probabilistic in nature (as is the case with genetic algorithms and simulated annealing), or the black-box simulation is stochastic, then it is important to be able to efficiently, accurately test whether one candidate is better than another (where a candidate can be, e.g., a metaheuristic configuration or a particular design). [1] have proposed several efficient methods for computing within given confidence bounds whether one candidate is better than another according to some utility measure; we are currently investigating:

- Extensions to these techniques,
- Integration of these techniques within our stochastic metaheuristics, and
- Applicability to our new formulation of adaptive problem solving.

Finally, we make use of parallelism and distributed processing on networks of workstations in order to provide the massive computational requirements of adaptive problem solving. Currently, processes are distributed using the Parallel Virtual Machines message passing package [5] at the black-box simulation level. For example, multiple copies of the black-box simulation are distributed and are executed in parallel given different decision variable assignments (i.e., a number of candidate designs are evaluated in parallel).

5. Example Spacecraft Design Optimization Problems

In this section, we describe two specific spacecraft design optimization problems to which we are currently applying the OASIS 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 range of different optimization problems that arise in spacecraft design.

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 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 micro-controller, a microtelecommunications system and a science payload package (a microlaser system for detecting subsurface water).

The optimization of physical design parameters for a soil penetrator based on these Mars microprobe is the first testbed for the OASIS system. The microprobe optimization domain in its entirety is very complex, involving a three-stage simulation:

- Separation analysis (i.e., separation from the Mars Surveyor),
- Aerodynamical simulation,
- Soil impact and penetration.

To illustrate the utility of adaptive problem solving, we now briefly describe current work on a simplified version of the soil penetration stage.

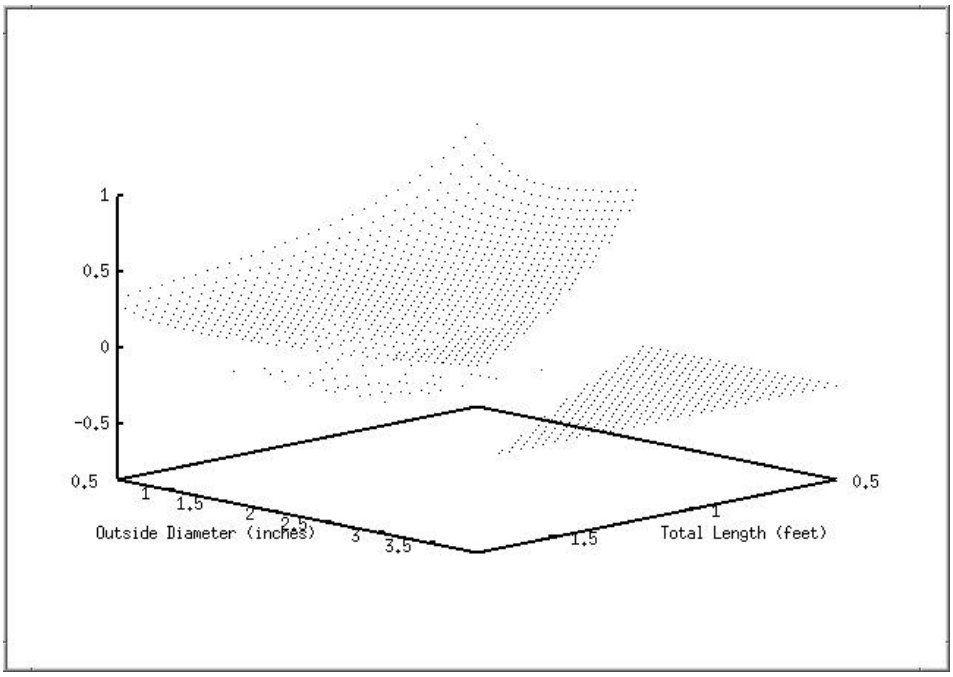

Figure 2—Sample points from cost surface for soil penetrator microprobe model. Plot of ratio of depth of penetration to length of penetrator. Soil number = 13 (soft soil)

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. We maximize this ratio, rather than simply maximizing the depth of penetration, since for the Mars microprobe science mission, the depth of penetration should ideally penetrate at least the length of the entire penetrator).

One of the initial condition parameters that has a significant impact on the structure of the cost surface for this optimization problem is the soil number, which indicates the hardness of the target surface. Intuitively, one would expect this to be an important parameter, since, for example, it is clearly more difficult to penetrate harder targets (the penetrator could bounce off the target, for example).

Figures 2 and 3 show plots of sample points from the cost surface of this simplified penetration problem for two different soil numbers, *soillNum=7* (hard), and *soilNum=13* (soft). The cost surface for *soilNum=13* is a relatively smooth surface, while the cost surface for *soilNum=7* is a much more rugged surface Because of the larger number of discontinuities in the cost surface for *soilNum=7*, optimization algorithms are more likely to get stuck in local maxima in this cost surface. We would expect that a greedy metaheuristic would be very successful for the soft surface, while a successful metaheuristic for the hard surface would require some mechanism to escape local minima. Therefore, to obtain the best performance on a similar problem instance (given a different soil number, for example), one should choose and configure a metaheuristic to exploit this knowledge ap-

propriately; this process would benefit from the application of our adaptive problem solving approach. The soil number is therefore a problem parameter that the OASIS adaptive problem solver can use as a feature with which to classify problem instances (i.e., into problems with soft and hard soil numbers).

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.) five 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 $400M in FY 94 dollars.

For the initial phase of the optimization demonstration, 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 260kg, the data volume output is zero. A spacecraft with a dry mass of greater than 260kg is too heavy to lift on the target launch vehicle. Thus the mass limit bounds 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

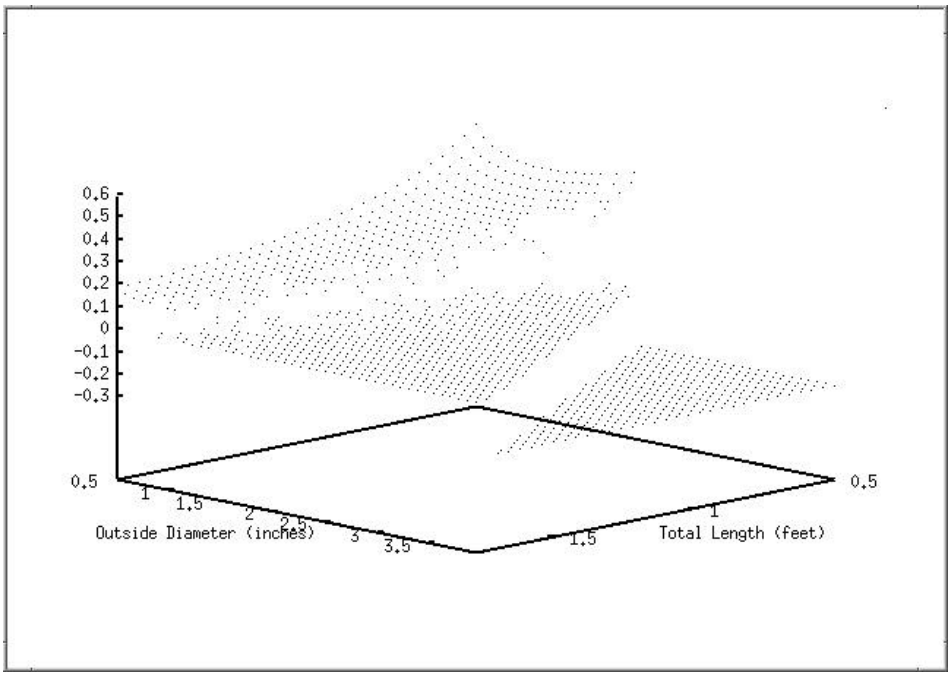

Figure 3—Sample points from cost surface for soil penetrator microprobe model. Plot of ratio of depth of penetration to length of penetrator. Soil number - 7 (hard soil).

spacecraft).

6. Summary and Conclusion

Designing a widely applicable tool for spacecraft design optimization is a significant technical challenge. In this paper, we have proposed the use of metaheuristic optimization algorithms, which are customized for particular problem instances by a process of adaptive problem solving. By this, we hope to provide a design optimization tool that can provide spacecraft designers with the ability to perform successful design optimization with minimal human effort. We have described OASIS, our current implementation of a system based on these principles, and discussed many of the technical issues that have arisen in its design. Adaptive problem solving for spacecraft design is a fertile research area with significant potential benefits; this paper has presented our initial efforts in this area.

Acknowledgments

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

References

[1] S. Chien, J. Gratch, and M. Burl. “On the efficient allocation of resources for hypothesis evaluation: A statistical approach,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):652-665, 1995.

[2] R. Dechter and J. Pearl. “Network-based heuristics for constraint-satisfaction problems,” *Artificial Intelligence*, 34(1):1-38, 1987.

[3] O. Etzioni. “Why Prodigy/EBL works,” In *Proceedings of National Conf. Artificial Intelligence (AAAI)*, 1990.

[4] D. Frost and R. Dechter. “In search of the best constraint satisfaction search,” In *Proceedings of National Conf. Artificial Intelligence (AAAI)*, 1994.

[5] 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.

[6] 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.

[7] F. Glover. “Tabu search - part I,” *Operations Research Society of America (ORSA) J. Computing*, 1:190-206, 1989.

[8] F. Glover. “Tabu search - part II,” *Operations Research Society of America (ORSA) J. Computing*, 2:4-32, 1990.

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

[10] J. Gratch and S. Chien. “Adaptive problem-solving for large-scale scheduling problems: A case study,” *Journal of Artificial Intelligence Research*, 4:365-396, 1996.

[11] J. Gratch and G. DeJong. “Composer: A probabilistic solution to the utility problem in speedup learning,” In *Proceedings of National Conf. Artificial Intelligence (AAAI)*, 1992.

[12] R. Greiner and I. Jurisca. “A statistical approach to solving the EBL utility problem,” In *Proceedings of National Conf. Artificial Intelligence (AAAI)*, 1992.

[13] D. Heckerman. “A tutorial on learning with Bayesian networks,” *Miscosoft Research Technical Report MSR-TR-95-06*, 1995.

[14] S. Kambhampati, C. Knoblock, and Q. Yang. “Planning as refinement search: A unified framework for evaluating design tradeoffs in partial order planning,” *Artificial Intelligence*, 66:167-238, 1995.

[15] S. Kirkpatrick, C. Gelatt, and M. Vecchi. “Optimization by simulated annealing,” *Science*, 220:671-680, 1983.

[16] J. Laird, A. Newell, and P. Rosenbloom. “Soar: An architecture for general intelligence,” *Artificial Intelligence*, 33(1):1-64, 1985.

[17] P. Laird. “Dynamic optimization,” In *Proc. Intl. Conf. Machine Learning*, 1992.

[18] S. Minton. *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, 1988.

[19] S. Minton. “Integrating heuristics for constraint satisfaction problems: A case study,” In *Proceedings of National Conf. Artificial Intelligence (AAAI)*, 1993.

[20] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[21] R. Smith and E. Smuda. “Adaptively resizing populations: algorithm, analysis, and first results,” *Complex Systems*, 9:47-72, 1995.

[22] P. Stone, M. Veloso, and J. Blythe. “The need for different domain independent heuristics,” In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, 1994.

[23] D. Subramanian and S. Hunter. “Measuring utility and the design of provably good EBL algorithms,” In *Proc. Intl. Conf. Machine Learning*, 1992.

[24] D. Wolpert and W. Macready. “The mathematics of search,” *Santa Fe Institute Technical Report SFI-TR-95-02-010*, 1994.

[25] M. Yagiura and T. Ibaraki. “Metaheuristics as robust and simple optimization tools,” In *Proc. IEEE International Conf. Evolutionary Computation*, 1996.

[26] S. Yakowitz and E. Lugosi. “Random search in the presence of noise, with application to machine learning,” *Society for Industrial and Applied Mathematics Journal of Scientific and Statistical Computing*, 11(4):702-712, 1990.