JOURNAL OF AIRCRAFT
Vol. 36, No. 1, January–February 1999

Conceptual Spacecraft Design Using a Genetic Algorithm Trade Selection Process

Todd Mosher*
The Aerospace Corporation, Los Angeles, California 90009-2957

Spacecraft design is a highly coupled problem. The design of the spacecraft must balance payload objectives and orbital design against cost and schedule guidelines. In commmon practice, a limited set of spacecraft design alternatives are considered and the design objectives are often not formally stated. With this approach there is no guarantee that a system-level focus will be taken, and feasibility rather than optimality is commonly all that is achieved. With support from NASA, the author has investigated better methods for performing the earliest stage of spacecraft design (including cost analysis) to ensure that a systems perspective is taken. This paper describes a tool for conceptual spacecraft design, the spacecraft concept optimization and utility tool (SCOUT), which uses a set of design-estimating relationships and cost-estimating relationships coupled with genetic algorithm optimization. A retrospective comparison between SCOUT results and the actual design of the Near Earth Asteroid Rendezvous spacecraft is made. This approach helps to enhance the understanding of the feasible design space, as well as uncovering interesting design points that merit further investigation.

Introduction

SPACECRAFT design is a difficult and challenging engineering problem. This is because it is difficult or impossible to retrieve a spacecraft if it fails to perform. The spacecraft also must survive severe launch and space environments. Designing a spacecraft is analogous to designing a car to run by remote control for 10 years at a distance of 22,000 miles without maintenance. Before the vehicle can be operated, it also must fold into a small volume and survive a rocket launch. Given these requirements, all that conceptual spacecraft designers are then asked to do is write the specifications and predict how much it will cost, sometimes five years in advance.

Despite these challenges, there have been 3810 successful space launches worldwide, and 2390 payloads were in orbit as of Dec. 31, 1996.[1] Although these numbers are dominated by the U.S. and the USSR/CIS, other nations have also launched spacecraft, including the European Space Agency consortium, China, Japan, Australia, Israel, and India.

Despite this wealth of space experience, the conceptual spacecraft design process is unstructured. The traditional conceptual design process can be represented by Fig. 1. As shown, a design group generates a set of requirements, concepts are synthesized by bringing all of the subsystems of a design together, an analysis of the candidate designs is performed, and a decision whether to advance with the current design is made. If the team determines the current design to be the *best* design, the results are documented in a conceptual design review. If a best design has not emerged, or if the team decides to study the problem more (resources allowing), the *change the design* loop is repeated. Conceptual design is typically an iterative process that will exercise the change to the design loop as often as resources will allow.

Interestingly, design researchers have found that actual design does not follow this process. Finger and Dixon[2] found in their survey of mechanical engineering design that designers often pursue a single design concept, patching and repairing their original idea rather than generating new alternatives. Conceptual spacecraft design also often suffers from this single-design concept fixation. This does not conform to the idea of true iteration, where fresh, creative ideas are generated each time through the loop.

To address this issue of generating a wide range of alternatives, a structured process and methods for improving conceptual spacecraft design using genetic algorithms is proposed.

Current Methods

Currently NASA and the spacecraft industry utilize three basic approaches to conceptual spacecraft design and cost. They are 1) parametric or empirical, 2) analogy, and 3) engineering buildup. The parametric or empirical design and cost approach, used by a majority of the industry, begins by gathering the experts in each subsystem including cost, and basically putting them into a room until a spacecraft is designed. Experts in each of the subsystem areas rely on their own parametric and empirical tools to design their respective area; a system engineer then tries to coordinate these efforts and to make them compatible. The analogy approach begins with the requirements and an existing spacecraft design. The existing design is then modified until the new set of requirements is met. Costs of the existing design are then adjusted to capture the differences. The engineering buildup design and cost approach derives the most essential subsystems from the requirements using catalogs of spacecraft parts. The design is iterated until all of the requirements are met and the subsystems defined. Costs are then derived from this list of equipment by using the catalog prices. All three of these methods explore a limited number of options (fewer than 10 is usually the limit because of schedule and cost constraints). Engineering buildup and analogy approaches tend to settle on a single point design very quickly.

Selection of a design and cost approach depends on the 1) scope of the effort, 2) detail of technical design definition, 3) availability and appropriateness of usable historical design and cost information, and 4) ability of the designers and cost estimators. Figure 2 shows notionally how the three design and cost methods are applied across program phases. The percentages used in the chart are estimates. The main point is the relative emphasis of each of the design and cost methods in each of the design phases.

Parametric or empirical approaches generally rely on mathematical expressions relating design and cost parameters as a

Received Oct. 26, 1997; revision received July 1, 1998; accepted for publication Aug. 9, 1998.

*Manager, P.O. Box 92957-M4/939. E-mail: todd.j.mosher@aero.org.

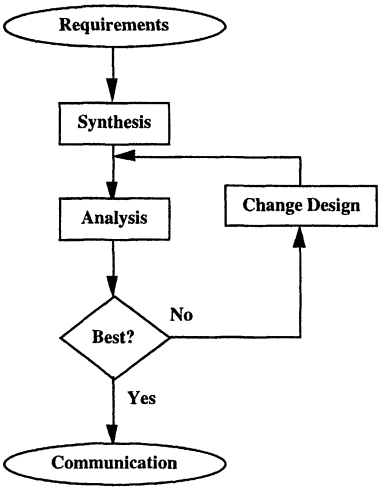

Fig. 1 Conceptual design process.[17]

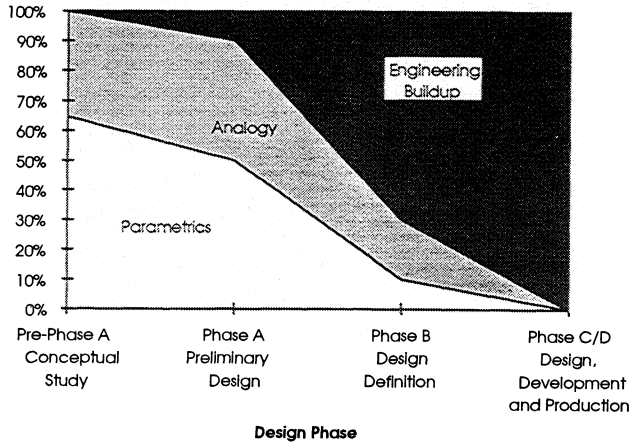

Fig. 2 Cost estimation by program phase (adapted from Ref. 3).

dependent variable to some other aspect of the spacecraft design. Both regression analysis, between data points for several existing systems that share characteristics with the system to be estimated, and engineering judgment, because pure statistical methods often fall short, are used to generate relationships called design estimating relationships (DERs) for design parameters and cost estimating relationships (CERs) for cost. A typical DER example is the structure subsystem's mass being a function of the total spacecraft mass to the supported. A typical CER example is relating the structure subsystem's cost to the structure subsystem's mass. An underlying assumption of the parametric approach is that the same forces that affected the design and cost of the systems that are used as data points will also affect the design and cost of the future system. Parametric methods can range from high-level rules of thumb to lower level models that use a set of subsystem DERs or CERs. Parametrics have become a standard method of estimating early program design parameters (such as spacecraft mass) and costs, but the assumption of using the past to predict the future that underlies this method is continually debated by engineers, managers, and cost analysts.

Analogous design and cost estimates are performed on the basis of comparison. Design parameters and costs from one or more past programs that are similar in complexity and objectives are used based on the analyst's judgment that those programs are representative of the program to be estimated. These design and cost data are then subjectively adjusted upward or downward to account for differences in the new program. This approach is used when there is sufficient knowledge of the new program and when there is comparable historical spacecraft design and cost data. A basic premise of the analogy approach is that no new system is actually totally new. In reality, most programs originate or evolve from already existing programs, representing simply a new combination of components or a modest evolution from past technologies. The advantage of the analogy approach is its relative simplicity when applicable information is available. The obvious disadvantages result from the difficulty of obtaining sufficient data for both the program to be estimated and the analogous programs.

Engineering buildup design and cost, also referred to as grass roots or bottom-up design, create design and cost estimates, by accumulating design parameters and costs from the lowest level of detail. To get an estimate by this method, the detailed spacecraft design, manufacturing, and procurement steps are laid out in great detail. Direct vendor bids are used for various components, and detailed labor costs are estimated. This analysis is aggregated according to the design to get an estimate with the greatest amount of detail possible. Engineering buildup estimates are often used in the latter phases of a project when the design configuration is relatively stable. However, they are also used when no historical design or costs are available and engineering judgment is the only source for the basis of the estimate. In this case, the validity of these designs and costs rests in the credibility of the expert opinion and in the fact that the design has been examined at a very low level. Engineering buildup has the advantage of being individually tailored to a specific program and is traceable to a very detailed level. Counter to this advantage is its requirements for volumes of information and documentation. It is also limited when the design has not reached a very detailed level. Flexibility is also a problem, because with each major program change a laborious new estimate must be generated. Finally, engineering judgment estimates can often be either inflated or overly optimistic, depending on the conservatism of the source.

In these three approaches there is no guarantee that a system level focus will be taken. Compromise is usually made at the systems level to get the subsystems to work together.[3] In the future, the emphasis will be on the development of highly efficient systems where overall system performance will be maximized and the compromises will be made in the subsystems. To obtain a complete understanding required for this to occur, better system level tools must be developed.[3]

All three design and cost estmating methods are used early in spacecraft design. Parametric methods are used primarily for comparison and the rapid evaluation of proposals, although there is some movement as their quality improves to make them an acceptable solution for generating a proposal design and cost estimate. Parametrics and some limited analogy methods are also used until a proposed design has been selected, because more detailed methods cannot respond to rapid changes. Analogy and engineering buildup are still the most common methods of generating the proposal cost estimate, with engineering buildup being limited by the amount of detailed design data generated at this early phase.

Changing Spacecraft Design Process

Relying on experience and intuition alone may be flawed because of the recent significant changes in the spacecraft design field. Commercial forces and scrutiny of government expenditures are raising expectations in spacecraft design. The new emphasis in spacecraft design is on economics, or getting *more bang for the buck*. Thus, the aerospace design practice has gone from maximizing performance under technology constraints to minimizing cost under performance constraints (Fig. 3). Out of this movement has emerged the slogan, *faster, better, cheaper*, meaning that spacecraft engineers must do more with less. The fast-paced schedules and small budgets characteristic of modern programs are forcing influential conceptual-stage decisions to be made more quickly by smaller teams. Modern budgetary and schedule constraints along with technology improvement and miniaturization have pushed spacecraft builders

toward smaller satellites. The trend toward smaller satellites is causing a revolution in how space systems are conceptualized, financed, procured, launched, and operated.[4]

NASA, and its Administrator Daniel Goldin, have wholeheartedly embraced the faster, better, cheaper slogan and the move to smaller spacecraft.[5,6] NASA has indicated its intent to emphasize the use of small spacecraft to perform the majority of its future space science and applications missions (Fig. 4, Ref. 7). This strategy is intended to result in a government space program with more frequent flights at markedly lower cost per flight. Because of the lower cost and increased frequency of flights, a higher tolerance for the risk associated with new technologies is expected. NASA believes this approach will also lead to shorter schedules, improved versatility, and an improved ability to transfer technology to the commercial industry.[6]

Along with the trend toward smaller satellites, new business approaches have made today's small spacecraft different than their similarly sized cousins of the past. Traditional large and small satellites have followed one curve of cost vs mass, whereas modern (post-1990) small spacecraft follow another curve (Fig. 5, Ref. 8).

Spacecraft Design as an Optimization Problem

Using the language of optimization in describing the spacecraft design process is not completely new. Individuals at the Jet Propulsion Laboratory,[9] and in academia[10] have used the concepts of an objective function, constraints, and design variables from the optimization field. Current spacecraft designs are optimized manually, using a tool-assisted evaluation of alternatives combined with consensus choices about design options and trades. Although a move toward more numerical optimization methods would seem natural, it has been resisted.[11]

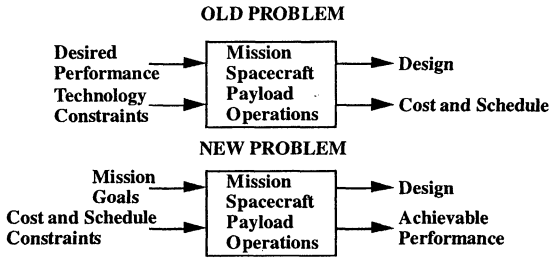

Fig. 3 Changing spacecraft design problem.

To meet this stated need, a revised version of the design methodology shown earlier in Fig. 1 is proposed. In Fig. 6, the requirements, synthesis, and change design steps are replaced with problem formulation, model construction, and automated search. Problem formulation broadens the requirements step to encompass defining the design problem in optimization terms and understanding the interaction between elements of the problem studied. Model construction involves not only performing the synthesis, but creating a model that captures it and the subsequent analysis step. Automated search takes advantage of advances in computer technology and mathematical search algorithms to search the design space. Using automated search removes human intuition-based bias and prevents the simple *patch and repair* approach through a broad, methodical computerized search. These new steps introduced into the design process have their roots in the field of operations research, and so this new design methodology is actually a hybrid of design and operations research approaches.

A general statement of a constrained optimization problem is presented next.

Objective function:	Minimize or maximize,	$f(\boldsymbol{x})$
Inequality constraints	subject to:	$g_j(\boldsymbol{x}) \leq 0, \quad j = 1, \ldots, p$
Equality constraints		$h_k(\boldsymbol{x}) = 0, \quad k = 1, \ldots, q$
Side constraints		$x_i^L \leq x_i \leq x_i^U$
Design variables	where:	$\boldsymbol{x} = \{x_1, x_2, \ldots x_n\}$
		$i = 1, \ldots, n$

An engineering system can be characterized by a set of quantities, some of which are viewed as variables during the design

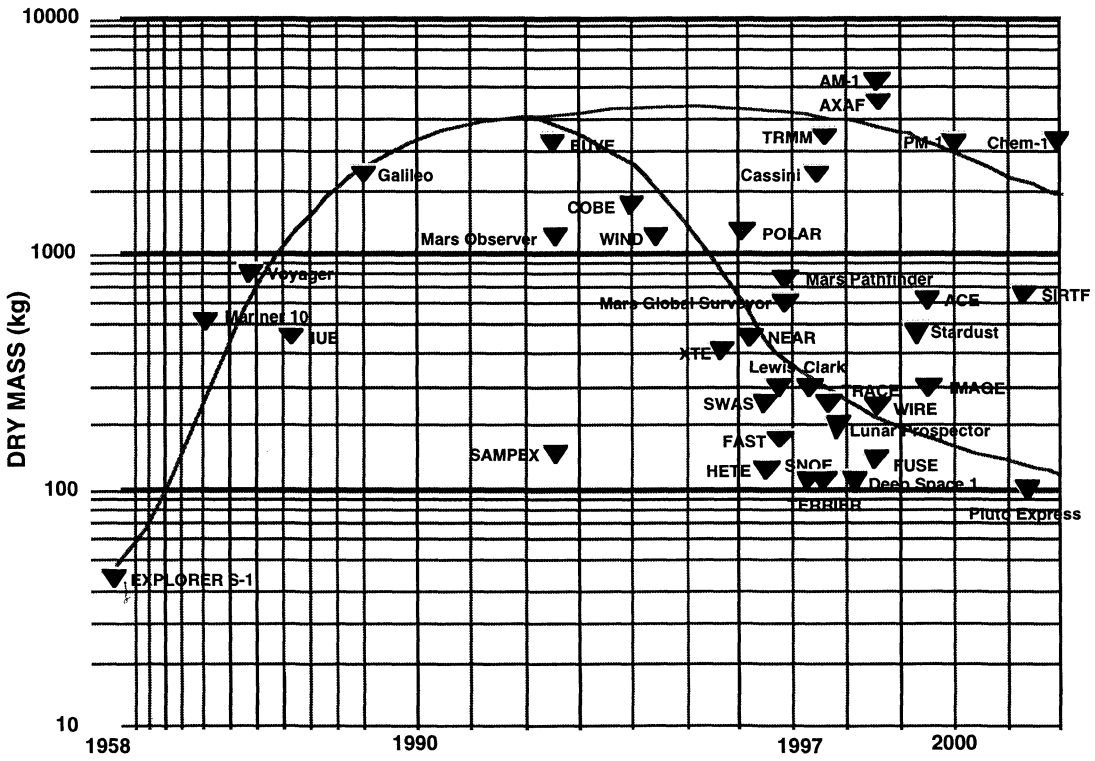

Fig. 4 NASA's move to small satellites.[6]

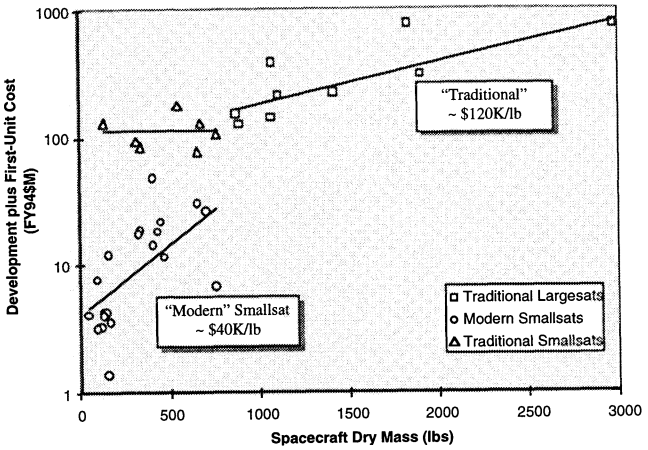

Fig. 5 Changes in cost for small satellites.[8]

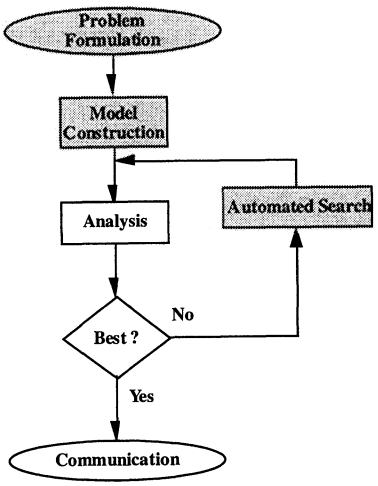

Fig. 6 Modified conceptual design process.

process. In general, some quantities are fixed at the outset and these are called *design inputs*. All of the other quantities that are allowed to vary in the design process can be called *design variables*, x_i, $i = 1, \ldots, n$. The design variables collectively are called a *design vector*.

In many problems, the design variables cannot take on all values and are said to be constrained. The restrictions on design variable values are called *design constraints* and are further characterized by their mathematical nature as *inequality constraints*, *equality constraints*, or *side constraints*. These constraints define the boundaries of the feasible and infeasible design space regions. Conceptual spacecraft design problems are generally constrained.

Conventional design is usually limited to finding the best feasible design from those few designs considered. When automated search is introduced into the design process, the aim is to find the best possible design of the many feasible designs that can be identified through a broad computerized search. Criteria must be chosen for comparing alternatives and selecting which are best. The *objective function* is the mathematical expression of the design goal. A design that optimizes (minimizes or maximizes) the objective function is, by definition, the best design. The choice of the objective function is governed by the nature of the problem and may even involve multiple criteria, in which case it is called a *multiple-objective function*.

Conceptual spacecraft design is a constrained problem involving nonlinear equations, with integer variables. The design variables are often discrete in nature with finite choices; thus, it fits the description of a combinatorial problem. In addition, for the model to work properly, the integer nature of the variables must be maintained so linear relaxation methods will not work. This classification of the problem eliminates most optimization approaches from consideration. Such methods as the simplex method, branch and bound, and many other mathematical programming techniques are not suitable for this problem.

Combinatorial problems are often easy to describe, but difficult to solve. For example, choosing the starting lineup for a baseball team is a combinatorial problem. For nine players, one can choose one out of the nine as the first batter. Then one of the remaining eight can be chosen for the second batter, one of the remaining seven for the third, and so on until a complete lineup is chosen. This eventually grows to 362,880 different lineups. If one uses a 20-person roster to fill the lineup, there are 60,949,324,800 possibilities. A constraint such as only one pitcher would reduce the problem, but it should be clear how quickly an easily described problem can grow.

An algorithm for solving an optimization problem is considered an *exact algorithm* if it is mathematically guaranteed to find an optimal solution if one exists. The many algorithms created for certain types of optimization problems such as linear, convex quadratic, and nonlinear convex programming problems are exact, because the optimality conditions providing efficient characterizations of the optimal solutions for these problems are known. However, there are many problems that do not allow for exact solutions, and often combinatorial optimization problems fall into this group.[12]

Many would say that today's computing power could handle enumeration of these large number of combinations. This is not always the case. Consider an example where all of the design variables are binary (restricted to a value of 0 or 1). In this case, the number of solutions grows at a rate of 2^n. Even a hypothetical computer capable of examining one billion design vectors per second, would require more than 30 years for $n = 60$, more than 60 years for $n = 61$, 10 centuries for $n = 65$, etc. However, specialized algorithms can, in most cases, solve a problem with $n = 100{,}000$ in a short time on a microcomputer.[13]

This problem characterization narrows the search for a method to a field of research entitled *meta-heuristics*. Meta-heuristics are the most recent development in approximate search methods for solving complex optimization problems. These methods have developed dramatically since their inception in the early 1980s, and have enjoyed widespread success on a variety of practical, yet difficult problems. Meta-heuristics include, but are not limited to, genetic algorithms, neural networks, simulated annealing, tabu search, and hybrids of these approaches using concepts derived from artificial intelligence, mathematics, and natural and physical sciences.[14]

Fortunately, the formulation of the conceptual spacecraft design problem chosen for this paper is of a size such that enumeration of all the possible solutions is feasible. However, as the previous discussion illustrates, with a slight increase in the options in the design vector, enumeration could quickly become impractical. Thus, it is important to explore another alternative to solving this problem. This author chose to focus on genetic algorithms to solve this problem. Genetic algorithms were selected because 1) genetic algorithms are well suited for combinatorial problems, 2) there exists genetic algorithm software that works easily with the spacecraft concept optimization and utility tool (SCOUT), and 3) genetic algorithms are a straightforward method that is easy to apply.

Genetic Algorithms

The first genetic algorithms were developed in the early 1970s by John Holland at the University of Michigan.[16] Holland was impressed by the ease with which biological systems could perform tasks that eluded even the most powerful supercomputers. These tasks included recognizing objects, understanding and translating sounds, and navigating through a dynamic environment. He began applying the properties of

Table 1 Comparison between natural selection and genetic algorithms[a]

Biological world	Genetic algorithm
There are many *organisms* in nature.	There are many possible *solutions* in a design space.
Each organism contains many *genes*.	Each solution is made up of *design variables*.
A group of organisms form a *population* where some of the organisms will be more fit than others.	A group of possible solutions can also be stored together as a *population* where some of the solutions will be *closer to optimal* than others.
Organisms that are more fit have more chances of *mating* and having *offspring*. These offspring are more likely to be better adapted because they received good genes.	Good solutions are selected more often to *combine their design variables* to form *new solutions*. The resulting solution is more likely to be better than a random guess because it is composed of good variables.
"Survival of the fittest" ensures that successful traits are passed along to subsequent generations and refined through evolution.	"Survival of the fittest" ensures that the schema of the best solutions is continually refined.

[a]Adapted from Ref. 15.

evolution to let number sets live, die, and breed like living organisms. Holland even borrowed the language of genetics to describe the methods of his algorithm,[15] as shown in Table 1. Originally applied to improve computer-programming structures and program performance, it has been realized that genetic algorithms have broader application potential. Only recently have they been classified as a meta-heuristic and applied to combinatorial problems.[14]

Genetic algorithms differ from calculus-based optimization methods in four major ways.

1) Genetic algorithms work with a coding of the design variables, not the design variables themselves.

2) Genetic algorithms search from a population of points instead of moving from a single point to the next.

3) Genetic algorithms need only a fitness function value. No derivatives or gradients are necessary.

4) Genetic algorithms use probabilistic transitions to find the next points in the search rather than deterministic transitions based on gradients.[16]

As alluded to in Table 1, genetic algorithms look at candidate design vectors as organisms and the design variables in the solution as *genes*. These genes are binary strings that represent the value of each of the design variables. A concatenated set of genes that represent a solution is called a *chromosome*. Genetic algorithms work with these chromosomes rather than with the actual design vector itself. This allows genetic algorithms to handle problems that simultaneously contain discrete, integer, and continuous (discretized to some resolution) design variables in a single problem.

Another distinguishing feature of genetic algorithms is that they work with a *population* of solutions rather than with a single solution. The fact that they evaluate multiple solutions simultaneously lends them to implementation via parallel computing. However, evaluating the entire population of each generation, as is commonly done, can also make them computationally expensive. This can be offset by a steady-state approach, where each time a new organism is created, it is inserted into the population and the lowest member is dropped. This is the approach taken on this problem.

To start the algorithm, an initial population is randomly generated by placing binary values along the chromosomes for the number of chromosomes in the population size. For example, 20 chromosomes would be created for a population of 20 organisms. Unlike the biological world, population size generally remains constant throughout the genetic algorithm execution. In a simple genetic algorithm, two parents are chosen to produce two *offspring* (this operator is called *mating*), so that the population size will remain the same. Population size has received a great deal of study in the genetic algorithm community, but a guideline that appears prudent is that the population size be at least four times larger than the string length of the chromosome.[17] Mating mimics biological sexual recombination for each parent's contribution of their DNA to the offspring. This is the main operator used to determine the next generation. The crossover rate is the proportion of the next

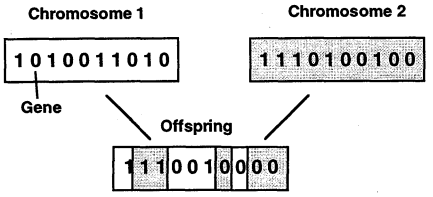

Fig. 7 Basic mating function of a genetic algorithm (adapted from Ref. 18).

generation's genes that is provided by each parent. If set at 0.50, a typical value, then each parent contributes 50% of their genes to the next generation's chromosome. When not equal (say 0.60) then the more fit parent will contribute a higher percentage of genes than the other parent. These basic concepts are illustrated in Fig. 7.

Analogous to the objective function, genetic algorithms use a fitness function to evaluate which members of the population are most fit. This fitness function is an evaluation of the design vector decoded from its chromosome representation to determine its fitness value. Design vectors with better fitness values are more likely to survive and be chosen as parents for the successive generation. This operator is called selection. There are many methods to perform selection, but all of them are probabilistic, making their results not exact or repeatable.

Along with mating and selection, the third operator used in genetic algorithms is *mutation*. Like the biological world, mutation happens infrequently in genetic algorithms (typical mutation probabilities are 1% or less). However, it does serve the important purpose of delaying the condition where a population would lose its diversity, making all of the points so similar that no further improvement is possible. The actual mechanics of mutation are simple; a gene in the chromosome is randomly flipped from 1 to 0 or vice versa. Despite the simplicity and infrequent occurrence of this operator being exercised, mutation's role in genetic algorithms is a hotly debated issue in the genetic algorithm research community. In summary, the genetic algorithm process used in this research is shown next. Description: 1) randomly generate initial population, 2) evaluate fitness of each individual, 3) select two parents, 4) mate for one offspring, 5) examine population for mutation, 6) evaluate fitness and add new offspring to population, and 7) eliminate lowest ranked individual and return to step 3. Genetic algorithms can be stopped after a predetermined time, when average fitness nears best fitness, when no further improvement occurs or when a known optimum is reached.

SCOUT Model

Sophisticated analysis tools have been developed to perform spacecraft modeling such as NASTRAN for structural analysis and SINDA for thermal analysis. Yet despite the presence of these highly sophisticated subsystem design tools, there have been limited efforts to enhance the capability of a system engineer in searching the vast, ill-defined design space. This

search still relies heavily on intuition-based qualitative design selections performed by the system engineer or the design team that he/she leads.

To address the need for a system engineering tool, SCOUT has been implemented in Microsoft Excel. SCOUT is a system-level model that captures the design and cost of the spacecraft as well as the performance and procurement of a launch vehicle. It is primarily focused on an interplanetary spacecraft design, but many of its approaches are applicable to Earth-orbiting spacecraft as well. SCOUT is not meant to be a model for wide application, but is a demonstration of this method of conceptual spacecraft design. This does not preclude the fact that this approach has widespread applicability, but use on a wider variety of problems has been left to future research.

SCOUT uses parametric design and cost approaches because it relies on previous experience to generate its many equations. Because these approaches rely on historical data, SCOUT only models current technology. This limits SCOUT in certain areas, such as power, where the focus is on electrical power rather than more exotic approaches like nuclear. The history used is recent (post-1990) spacecraft, and so the choice of historical database has been made to reflect modern spacecraft. The method by which spacecraft subsystems are modeled is through the creation of parametric relationships that relate known or previously calculated variables to performance parameters such as mass or cost. These relationships are referred to as DERs when they are calculating design parameters like mass and CERs when they are calculating cost.

Because of Excel's object-oriented hierarchy, it was necessary to construct SCOUT in a single spreadsheet within a single workbook so that the optimization routines would operate properly. Future work will be performed to implement SCOUT in a multiple worksheet model because this is the current form of many Excel-based spacecraft design tools. Given that limitation, Fig. 8 shows that SCOUT contains three basic areas: the graphical user interface (GUI), the design and cost calculations, and the optimization portion. The design and cost calculations are further broken down by topic area or subsystem, ranging from mission to system.

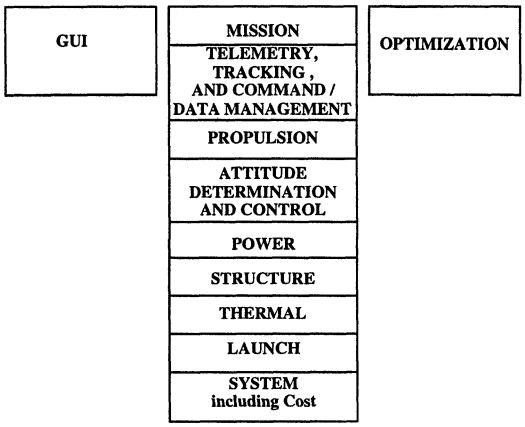

Fig. 8 SCOUT layout.

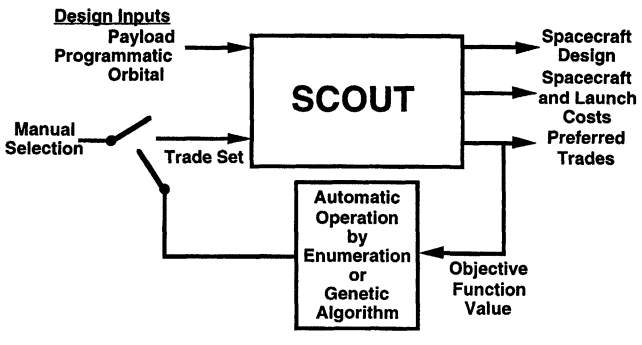

Fig. 9 SCOUT operation.

The operation of SCOUT is shown in Fig. 9. SCOUT is driven by six primary trades that constitute the design vector. These are listed in Table 2 along with their options. This trade set was chosen to capture the major trades performed at this phase of design, and so that enumeration would be possible in this example. This list was arrived at after studying many different spacecraft proposals as well as consulting with many in the spacecraft community. Other trades can be considered, but these were judged the most important. SCOUT also requires several design inputs about the payload, programmatic, and orbital characteristics of the mission being studied. These are listed in Table 3 along with the actual values used for the Near Earth Asteroid Rendezvous (NEAR) example.

The SCOUT GUI (Fig. 10) allows the user to see the main characteristics of the spacecraft design problem, the design vector of selected trades, and provides buttons that control recalculation, enumeration, and the genetic algorithm. The trade set options may be selected manually by the user from the SCOUT GUI or can be chosen automatically by either the enumeration or genetic algorithm schemes. SCOUT calculates the values of 77 parameters that it uses to generate a conceptual spacecraft design defined in terms of mass and power breakdowns. SCOUT also produces spacecraft and launch costs, and a preferred set of trades that provide further subsystem definition. Figure 11 shows how the various modules within SCOUT interact.

Generating a complete set of solutions can be time consuming or impossible. However, for a problem of sufficiently small number of combinations, it can be worth the wait. One can then search this solution set and be guaranteed of finding the exact optimum and learn much about the nature of the design

Table 2 SCOUT trades and options

Trades	Options
Cell type	Silicon or gallium arsenide
Array type	Body mounted, fixed-deployed array, or Sun tracking-deployed array
Battery type	Nickel cadmium or nickel hydrogen
Structure type	Aluminum or composites
Propulsion type	Solid/mono-prop or dual mode bi-prop
Launch vehicle type	Atlas IIAR, Atlas IIAS, Atlas IIA, Delta 7925-H 9.5, Delta 7925 9.5, Delta 7325-9.5, Athena II, SELV II-B, SELV II-A

Table 3 SCOUT NEAR design inputs

Parameter	Units	NEAR values
Payload mass	kg	56
Payload power	W	93.5
Payload eclipse power	W	93.5
Pointing knowledge	deg	0.097
Pointing accuracy	deg	0.029
Downlink datarate	kbps	26.5
Year of launch	yr	96
Design life	mo	48
Mass margin	%	0
Power margin	%	0
Destination	string	Eros
Launch energy (C3)	km^2/s^2	25.9
Period	min	90
Eclipse duration	min	30
Communication during eclipse	Boolean	TRUE
Maneuver and attitude control Δ velocity	km/s	0.125
Solar flux	W/m^2	432
Time at destination	mo	11
Orbit insertion Δ velocity	km/s	1.275
Destination distance from sun	AU	1.78
Destination body radius	km	40
Sun angle	deg	30

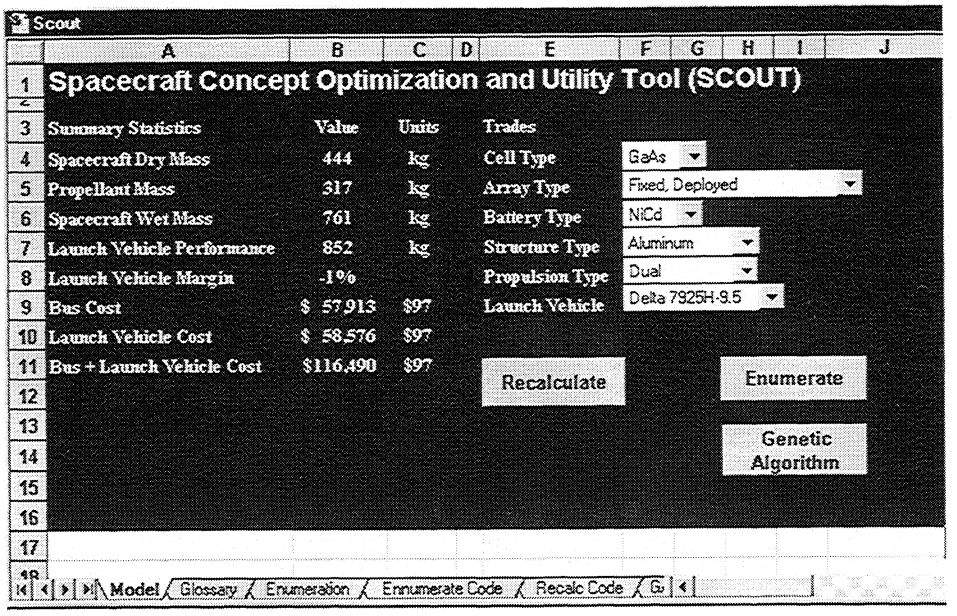

Fig. 10 SCOUT GUI.

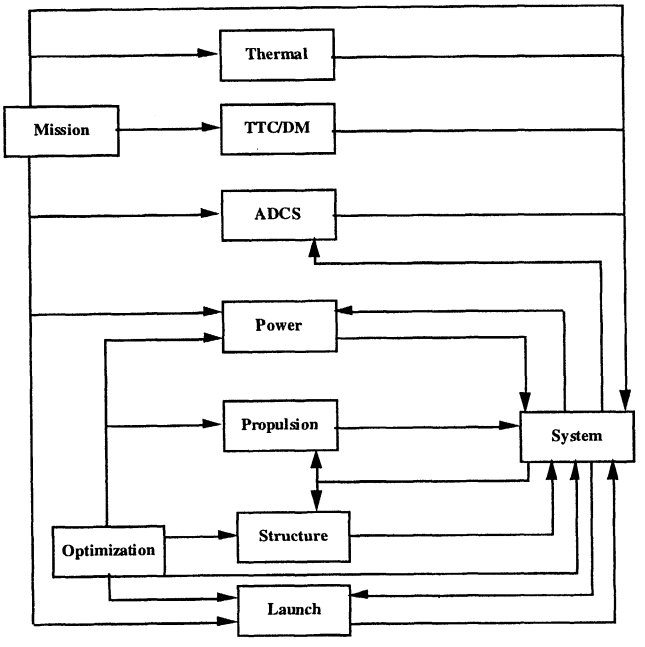

Fig. 11 Flow of information in the SCOUT model.

space in the process. Enumeration can provide the definitive answer to moderate sized spacecraft design problems. Enumeration is implemented in SCOUT by creating a nested loop structure in Visual Basic for Applications, the standard Microsoft scripting language.

The genetic algorithm approach to the spacecraft design problem is implemented into Excel using the Excel add-in Evolver.[18] The genetic algorithm fitness function for this problem is set to minimize the combined bus and launch vehicle cost subject to a penalty function associated with the launch vehicle margin and a feasibility of body mounting if this is a trade option selected. The design variables are the various spacecraft trades that are constrained by side constraints based on the number of options available for each trade. The design variables are also constrained to integers so that the logic built into SCOUT operates correctly.

Body mounting feasibility is modeled as a hard constraint in SCOUT. It consists of a determination of whether there is sufficient surface area on the spacecraft to body mount the necessary array area needed to supply spacecraft power. This is modeled via Boolean logic with a TRUE occurring when this trade option of a body mounted solar array is not selected (and, hence, the constraint is not applicable), or when necessary surface area is available for the array mounting when the body mounted option is chosen. If this constraint cell evaluates to FALSE, Evolver attempts to modify the infeasible solution so that it becomes feasible by a built-in procedure called *backtracking*. This makes it more difficult and computationally more expensive for Evolver to handle hard constraints. However, in this case, there is no clear way to do this differently.

While the launch margin can also be modeled as a hard constraint (must be greater than zero), this research found it was better to model it as a soft constraint. A soft constraint uses a penalty function that grows in effect as the constraint is violated. This allows Evolver to explore more widely and even arrive at solutions that are a slight violation of a constraint, yet may be dramatically better than another solution that does not violate this constraint even when the penalty is included. Also, at this stage in the design, all of the relationships have some sort of error inherent in them, and so, it does not make sense to throw out solutions that violate this constraint, unless it does so significantly. Therefore, SCOUT models launch margin as a soft constraint using a penalty function that grows linearly as it is violated.

In SCOUT, the genetic algorithm parameters are set at a population size of 50, a crossover rate of 0.50, and a mutation rate of 0.06, which are the recommended values for a problem of this type.[17,18] Experimentation with different parameter settings was not performed in this example.

SCOUT Compared with NEAR

A systematic approach to testing a model is to use a retrospective test. This test involves using historical data to reconstruct a past program and then determine how well the model does at recreating the known solution. Comparing the hypothetical performance of the model with reality then indicates if the model emulates or even improves upon existing practice. It may also indicate areas where the model has shortcomings and needs improvement. One disadvantage of using a retrospective test is the issue of whether the past is representative of the future. That is why the past that is embedded in the model must be carefully chosen.

The NEAR spacecraft is chosen for comparison because it was recently recognized with a Laurel award by *Aviation Week*

and Space Technology as a benchmark for NASA's Discovery program.[19] The Discovery program is designed to accomplish high-quality, focused planetary science missions that substantially reduce total mission cost while improving performance. It also seeks to enhance public awareness and appreciation of planetary exploration. Discovery missions are constrained in funding, launch options, and schedule. Phase C/D (development/production) spending is limited to $150M (FY92$), and Phase E (mission operations) costs are capped at $35M (FY92$). Launch must occur on a Delta II or smaller launch vehicle. The schedule is limited to 18 months for Phase A/B (concept development) and 36 months for Phase C/D.[20] The NEAR mission is the second Discovery spacecraft and the first to be launched. Its primary mission is to rendezvous and orbit with the asteroid 433 Eros. Launched on Feb. 17, 1996, it is depicted in Fig. 12.[21]

SCOUT was initially operated using the enumeration approach with Delta as the largest launch vehicle. What is interesting about the NEAR design is that because it is such a constrained problem, there are very few solutions that are feasible if the Discovery constraint of a Delta launch is applied. In fact, only one option shows positive margin, and only four solutions are near a positive margin (within 2%). Holding only to solutions with a positive margin, as indicated by the SCOUT model, is perhaps too rigid because of its inaccuracies and because there may be some flexibility available in the real design. The actual NEAR design showed a negative margin when modeled in SCOUT, so this raised a flag for further investigation.

After further research,[21] it was found that the NEAR project made several modifications to the baseline Delta configuration. These included the use of high-performance nozzles, a smaller payload fairing, and a lightweight payload attach fitting. So, in fact, the actual NEAR did not use one of the launch options modeled in SCOUT, but rather a custom option driven by being a highly constrained problem.

NEAR was originally modeled with the Delta as the highest-capability launch option, but given the resulting limited feasible design space, the problem was expanded with three higher-capability Atlas launch vehicle options as well as an additional smaller vehicle option, the Athena (formerly known as the Lockheed Martin Launch Vehicle). This automatically expanded the feasible design space to 125 solutions. However, all of the combinations using the Atlas cost considerably more than the four identified Delta options, whereas none of the Athena options were feasible. Given this trade-space exploration, the four options shown in Table 4 were highlighted for significance. Among the four options was the one performed earlier that matched the SCOUT set to NEAR's trade options (Solution-1). Most notable is that the other three solutions also meet a ±2% launch margin tolerance. These suggest that there are several solutions that could prove to be the optimum.

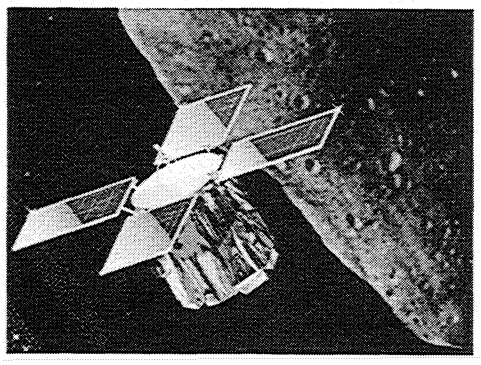

Fig. 12 NEAR spacecraft at Eros.

While Solution-1 indicates the lowest spacecraft bus plus launch vehicle cost, inaccuracies within SCOUT (part of the reason for the differences with the NEAR actual costs) prevent stating conclusively that this is the optimum. In fact, finding the exact optimum is probably not really the point of the search. Rather, the combinations exhibited in these four options should be compared to determine the significant trades for further study. SCOUT already indicates that the choices for cell type, array type, propulsion type, and launch vehicle are uniform across all of these promising combinations. Thus, a design team should focus on what's not common (battery and structure types) as the primary trades to be further studied for the most promising solution.

Now given that for this problem, enumeration provided the optimal solution, why use the genetic algorithm? Well, as explained before, an enlargement of the trade set could quickly make enumeration an impractical approach. So it is important to see how the genetic algorithm used in SCOUT performed.

Because, by their nature, genetic algorithms can be thought of as a random process, it is important to run a Monte Carlo study to characterize the performance of the genetic algorithm. A simple trial of 100 runs of Evolver was performed with the genetic algorithm parameter settings as described previously and with the same starting population. Each run of Evolver took between 2 and 4 min on the personal computer used, so that this study took several hours to perform.

The majority of computer time for both the enumeration and the genetic algorithm is used in evaluating the objective function in SCOUT rather than any code specific to the search method. Therefore, comparing evaluations of SCOUT rather than absolute time is probably the best metric for comparison. The computers used in this analysis also varied in speed because of hardware differences, so that it is felt that number of SCOUT evaluations is a better metric for comparison. The total number for enumeration is 432 evaluations, so this sets an upper limit. The genetic algorithm was allowed to proceed to 500 evaluations to try and reach the optimum. If it did not reach the optimum, the trial was deemed a failure, and the best result found was recorded.

In only six of the 100 runs did the genetic algorithm fail to find one of the four solutions (solution-1 through solution-4) in Table 4 within the 500 evaluations chosen as the stopping condition for the Monte Carlo analysis (because a genetic algorithm can literally run forever). Of the remaining 94 runs, it settled on the optimum (solution-1) 60 times, solution-2, 6 times, solution-3, 22 times, and solution-4, 6 times. Even when it failed, it did not settle on an infeasible solution, but instead converged to one of the higher cost, feasible Atlas combina-

Table 4 NEAR optimal SCOUT solutions

Design variables	NEAR Actual	Solution-1	Solution-2	Solution-3	Solution-4
Cell type	GaAs	GaAs	GaAs	GaAs	GaAs
Array type	Fixed, deployed	Fixed, deployed	Fixed, deployed	Fixed, deployed	Fixed, deployed
Battery type	NiCd	NiCd	NiCd	NiH	NiH
Structure type	Aluminum	Aluminum	Composites	Aluminum	Composites
Propulsion type	Dual	Dual	Dual	Dual	Dual
Launch vehicle	Delta 7925-H[a]	Delta 7925-H	Delta 7925-H	Delta 7925-H	Delta 7925-H
Objective function (FY97$M)	125.1	116.5	118.7	119.5	121.8
Launch margin constraint (%)	——[a]	−1.4%	−0.3%	−0.8%	0.2%

[a]NEAR actually flew a modified Delta not reflected in SCOUT. For this table, Delta 7925-H costs were used.

tions. When the genetic algorithm failed to find the optimum and settled on another solution, the cause was always loss of the gene needed to make it optimal. For instance, when Evolver settled for solution-2, then the aluminum gene had been lost through the mating process, even though it contributed to the optimal solution. Only mutation could bring it back into the gene pool, which theoretically might occur if the genetic algorithm was allowed to run for greater than 500 evaluations.

When the genetic algorithm did find the optimal solution, the mean time to find it was 212 evaluations (meaning that the genetic algorithm found the solution in less than half the number of SCOUT evaluations than the enumeration) with a sample standard deviation of 89. This mean time to find the optimum is based on the second stopping criterion of no greater than a 0.0001 improvement in the last 100 evaluations. This stopping criterion means that the optimal organism was actually found 100 evaluations earlier, but 100 more evaluations were run after that to ensure that it was optimal.

Conclusions

By using automated search methods, a significant increase in trade space exploration over the human intuition method (usually at best 10 options) was achieved. Even if it is assumed that a human designer or team could rule out all of the infeasible solutions, exploring a greater number of feasible options is possible using SCOUT and still is an order of magnitude beyond the conventional manual approach. This automated search of the design space is also unbiased, allowing it to identify counterintuitive solutions. This research presents a general approach that can be applied to other space-system problems and possibly even to design problem areas other than space systems.

Because the problem as formulated was small enough in the number of combinations needed to capture a complete set (432), enumeration could be performed and compared with the performance of a meta-heuristic method, genetic algorithms. Understanding that problem expansion can render enumeration impractical, the use of a meta-heuristic may be a long-term solution.

There is a price to be paid for the improved efficiency of genetic algorithms in that they are not exact. However, if solving close to optimum, i.e., finding one of the set of promising solutions identified, is acceptable, genetic algorithms are successful most (~95%) of the time based on this demonstration. The product of this approach is at least a high-quality seed design, or group of designs, around which further analysis can be performed.

Genetic algorithms' opportunistic behavior and parallel design pursuit make them an interesting mimic of the same behaviors that have been called favorable characteristics of human iteration. Thus, in some sense, by exhibiting some form of *creativity*, genetic algorithms may do much better than some other algorithm approaches at capturing what humans would do if they could perform with the same computational power.

Whether enumeration or genetic algorithms are used as the automated search algorithm, this improved design approach does appear to demonstrate a faster design process that could yield cost savings by removing humans from the task of iteration. It moves spacecraft conceptual design forward from its already achieved ability to perform Pre-Phase A studies in weeks, rather than months, toward the possibility of days.

Acknowledgments

The author would like to recognize his advisor George W. Morgenthaler, from the Aerospace Engineering Sciences Department at the University of Colorado, for his guidance. This paper is based upon a dissertation supported by a NASA Graduate Student Researcher's Fellowship funded by NASA Headquarters. It also has been supported by The Aerospace Corporation.

References

[1]Thompson, T. D. (ed.), *TRW Space Log 1996*, TRW, Redondo Beach, CA, 1997.

[2]Finger, S., and Dixon, J. R., "A Review of Research in Mechanical Engineering Design Part I: Descriptive, Prescriptive, and Computer-Based Models of the Design Processes," *Research in Engineering Design*, Vol. 1, No. 1, 1989, pp. 51–67.

[3]Casani, K., "Reengineering the JPL Project Design Process," Jet Propulsion Lab., Pasadena, CA, Aug. 1994.

[4]Abramson, R. L., Bearden, D. A., and Glackin, D. L., "Small Satellites: Cost Methodologies and Remote Sensing Issues," *SPIE/CNES European Symposium on Satellite Remote Sensing*, Sept. 1995.

[5]David, L., "Faster, Better, Cheaper: Sloganeering or Good Engineering?" *Aerospace America*, Jan. 1995, pp. 28–33.

[6]*Technology for Small Spacecraft*, National Research Council, National Academy Press, Washington, DC, 1994.

[7]Sarsfield, L., *Federal Investments in Small Spacecraft*, The RAND Corp., DRU-1494-OSTP, Washington, DC, Sept. 1996.

[8]Bearden, D. A.," Smaller, Better, Faster—And How Much Does it Cost?" *9th Annual AIAA/USU Conference on Small Satellites*, Sept. 1995.

[9]Shishko, R., and Jorgensen, E. J.," Design-to-Cost for Space Missions, *Reducing Space Mission Cost*, edited by J. R. Wertz and W. J. Larson, Microcosm Press, Torrance, CA, 1996, pp. 229–252.

[10]Morgenthaler, G. W., "Application of Binary Linear Programming to the Optimal Selection of Experiments for a Space Exploration Payload," Univ. of Colorado Center for Space Construction, TN 136, Boulder, CO, Aug. 1993.

[11]Mosher, T. J., "Applicability of Selected Multidisciplinary Design Optimization Methods to Conceptual Spacecraft Design," *Proceedings of the AIAA/NASA/ISSMO 6th Symposium on Multidisciplinary Analysis and Optimization* (Seattle, WA), 1996, pp. 664–671.

[12]Murty, K. A., *Operations Research: Deterministic Optimization Models*, Prentice–Hall, Englewood Cliffs, NJ, 1995.

[13]Martello, S., and Toth, P., *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, New York, 1990.

[14]Osman, I. H., and Kelly, J. P., *Meta-Heuristics: Theory and Applications*, Kluwer Academic, Norwell, MA, 1996.

[15]Kennedy, S., "Genetic Algorithms: Digital Darwinism," *Hitchhiker's Guide to Artificial Intelligence*, Miller Freeman, San Francisco, CA, June 1995, pp. 29–36.

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

[17]Crossley, W. A., "Genetic Algorithms for Engineering Design and Optimization: An Introduction," The Aerospace Corporation, El Segundo, CA, March 1997.

[18]Kennedy, S., *Evolver User's Guide*, Axcelis, Inc., Seattle, WA, 1996.

[19]North, D. M. (ed.), "Laurels," *Aviation Week and Space Technology*, April 7, 1997, p. 11.

[20]"Announcement of Opportunity for the Discovery Program," NASA, AO-96-OSS-02, Washington, DC, Sept. 1996.

[21]Farquhar, R. W. (ed.), "NEAR Spacecraft and Instrumentation," *The Journal of the Astronautical Sciences: Special Issue on the Near-Earth Asteroid Rendezvous Mission*, Vol. 43, No. 4, 1995, pp. 349–397.