S52-61

150522

N93-25610

A GENETIC ALGORITHM TOOL (SPLICER) FOR COMPLEX SCHEDULING PROBLEMS AND THE SPACE STATION FREEDOM RESUPPLY PROBLEM

Lui Wang
Software Technology Branch
NASA Johnson Space Center
Houston, Texas

Manuel Valenzuela-Rendon
ITESM, Campus Monterrey
Center for Artificial Intelligence
Monterrey, N.L., Mexico

Abstract

The Space Station Freedom will require the supply of items in a regular fashion. A schedule for the delivery of these items is not easy to design due to the large span of time involved and the possibility of cancellations and changes in shuttle flights. This paper presents the basic concepts of a genetic algorithm model, and also presents the results of an effort to apply genetic algorithms to the design of propellant resupply schedules. As part of this effort, a simple simulator and an encoding by which a genetic algorithm can find near optimal schedules have been developed. Additionally, this paper proposes ways in which robust schedules, i.e. schedules that can tolerate small changes, can be found using genetic algorithms.

1. Introduction

A schedule for the delivery of resupplies to the Space Station Freedom is not easy to design due to the large span of time involved and the possibility of cancellations and changes in shuttle flights. Additionally, there is difficulty in defining ways to determine the quality of schedules, many factors should be optimized. Genetic algorithms seem adequate for the task of finding appropriate schedules due to their proven ability to deal with complex objective functions and their robustness as search methods.

An initial effort to study the application of genetic algorithms to finding schedules for the resupply of propellant to the space station has been undertaken at the Software Technology Branch of NASA Johnson Space Center. As a result of this effort, a simulator was developed. A representation of schedules was designed. Also, an evaluation mechanism was proposed.

This paper first briefly describes the basic concepts of a genetic algorithm model, followed by a short description of the design of a genetic algorithm tool (Splicer) which was developed by the Software Technology Branch. Finally, the paper describes the results of the effort to apply genetic algorithms to the design of schedules for the resupply of propellant to the Space Station Freedom.

2. Genetic Algorithms and Splicer

2.1 Genetic Algorithms

Genetic algorithms (GA) are highly parallel, mathematical, adaptive search procedures based loosely on the processes of natural genetics and Darwinian survival of the fittest. These algorithms apply genetically-inspired operators to populations of potential solutions in an iterative fashion, creating new populations while searching for an optimal (or near-optimal) solution to the problem at hand. There are several key features of this search and optimization technique. One, the problem space is searched in parallel based on the "building blocks" concept. Two, genetic algorithms are very effective when searching function spaces that are not smooth or continuous–functions which are very difficult (or impossible) to search using calculus based methods. Three, genetic algorithms are blind: that is, they know nothing of the problem being solved other than payoff or penalty (i.e., objective function) information.

The basic iterative model of the genetic algorithms is: the algorithm starts up with an random population, and subsequent populations are created from the previous population by means of *evaluation, selection*, and *reproduction*. This process repeats itself until the population converges on an optimal solution or some other stopping condition is reached.

The initial population consists a set of individuals (i.e., potential solutions) generated randomly or heuristically. In the classical genetic algorithm, each member is represented by a fixed-length binary string of bits (a *chromosome*) that encodes parameters of the problem. This encoded string can be decoded to give the integer values for these parameters.

Once the initial population has been created, the evaluation phase begins. The genetic algorithms require that members of the population can be differentiated according to *goodness* or *fitness*. The members that are more fit are given a higher probability of participating during the selection and reproduction phases. Fitness is measured by decoding a chromosome and using the decoded parameters as input to the objective function. The value returned by the objective function (or some transformation of it) is used as the fitness value.

During the selection phase, the population members are given a target sampling rate which is based on fitness and determines how many times a member will mate during this generation–that is, how many offspring from this individual will be created in the next population. The target sampling rate (usually not a whole number) must be transformed into an integer number of matings for each individual. There are many ways of determining the target sampling rate and the actual number of matings. Suffice it to say that individuals that are more fit are given a reproductive advantage over less fit members.

During the reproduction phase, two members of the mating pool (i.e., members of the population with non-zero mating counts) are randomly chosen and genetic operators are applied to their genetic material to produce two new members for the next population. This process is repeated until the next population is filled. The recombination phase usually involves two operators: crossover and mutation. During crossover, the two parents exchange substring information (genetic material) at a random position in the chromosomes to produce two new strings. Crossover occurs according to a crossover probability, usually between 0.5 and 1.0. The crossover operation searches for better *building blocks* within the genetic material which combine to create optimal or near-optimal problem parameters and, therefore, problem solutions, when the string is decoded. Mutation is a secondary operation in the genetic algorithm process. It is used to maintain diversity in the population–that is, to keep the population from prematurely converging on one solution and to create genetic material that may not be present in the current population. The mechanics of the mutation operation are simple: for each position in a string created during crossover, change the value at that position according to a mutation probability. The mutation probability is usually very low–less than 0.05.

2.2 Splicer

The Splicer tool is a project within the Software Technology Branch. The purpose of the project is to develop a tool that will enable the widespread use of genetic algorithm technology.

The design chosen for the Splicer consists of four components: a genetic algorithm kernel and three types of interchangeable libraries or modules: representation libraries, fitness modules, and user interface libraries.

A *genetic algorithm kernel* was developed that is independent of representation (i.e., problem encoding), fitness function, or user interface type. The GA kernel comprises all functions necessary for the manipulation of populations. These functions include the creation of populations and population members, the iterative population model, fitness scaling, parent selection and sampling, and the generation of population statistics. In addition, miscellaneous functions are included in the kernel (e.g., random number generators). Different types of problem-encoding schemes and functions are defined and stored in interchangeable *representation libraries*. This allows the GA kernel to be used for any representation scheme. At present, the Splicer tool provides representation libraries for binary strings and for permutations. These libraries contain functions for the definition, creation, and decoding of genetic strings, as well as multiple crossover and mutation operators. Furthermore, the Splicer tool defines the appropriate interfaces to allow users to create new representation libraries (e.g., for use with vectors or grammars).

Fitness functions are defined and stored in interchangeable *fitness modules*. Fitness modules are the only piece of the Splicer system a user will normally be required to create or alter to solve a particular problem. Within a fitness module, a user can create a fitness function, set the initial values for various Splicer control parameters (e.g., population size), create a function which graphically draws the best solutions as they are found, and provide descriptive information about the problem being solved. The tool comes with several example fitness modules.

The Splicer tool provides three *user interface libraries*: a Macintosh user interface, an X Window System user interface, and a simple, menu-driven, character-based user interface. The first two user interfaces are event-driven and provide graphic output using windows.

The C programming language was chosen for portability and speed. Splicer has been tested on multiple platforms which include Sun 3/80™, SPARC™, IBM RS6000™, and Apple Macintosh™. With the new character and TTY interfaces, Splicer can now be embedded in the user application.

3. The Space Station Freedom scheduling problem

The Space Station Freedom will require the supply of various items in a regular fashion, including such things as air, food, experiment payload modules, and propellant. A schedule for the delivery of these items spanning several years will be needed. Because of the large number of activities to be scheduled, and the long period of time involved, it is not possible to plan them all at the same time in a single schedule. Instead, independent schedules for separate items will be designed. Then, these will be merged into a single schedule. To facilitate this integration, each individual schedule must be as flexible and robust as possible, i.e. changes must be easily made and must not seriously affect the overall performance. Additionally, the overall schedule must also be robust so that shuttle flight delays and cancellations can be tolerated without major changes. Genetic Algorithms (Goldberg, 1989, Holland, 1975), due to their inherit flexibility, seem to be an appropriate tool for solving this problem because the complexity of the many restrictions that can be involved in schedules for individual items.

As a first step in studying the applicability of genetic algorithms, the problem of scheduling the resupply of propellant was selected. The following description of propellant resupply to the Space Station Freedom was a result of several talks with the Level II Space Station Freedom Resource Utilization Analysis engineers.

3.1 Description of the propellant resupply problem

Reboosting at the space station occurs after every departure of the space shuttle. The reboost operation takes the space station to its highest orbit. Between reboosts, the space station slowly looses altitude, so as to meet the shuttle at its lowest orbit.

The thrust required for a reboost operation is supplied by three out of six reboosting modules. These modules contain a propellant which is consumed to produce a force on the space station. The space station has eight parking spaces where reboosting modules can be placed. These eight spaces are grouped in pairs and each pair is located on a corner of an imaginary rectangle perpendicular to the Earth. There are a total of eight modules; at any given time, six modules are expected to be on the space station and two on ground.

For any reboost operation to be carried out, three modules in different corners must be fired; in this way, a force and a torque, in any desired direction perpendicular to the Earth, can be applied to the space station. For a given reboost, and assuming that the modules contain sufficient propellant, there will always be two sets of three corners of the rectangle to choose from, i.e. two triangles (call them A and B) that can produce the same force and torque. Both triangles will be able to produce the same reboosting effect, and require the same total propellant, but will consume different levels of propellant from the individual corners involved. The level of propellant required for a given reboost operation is not constant; it depends on the solar activity at the moment, and thus, can only be estimated ahead of time.

The space station will normally have five reboost cycles per year, one for each flight of the space shuttle. At its lowest, orbit it will meet the space shuttle and receive resupplies, then it will be reboosted to its highest orbit. The resupply of propellant will normally consist of delivering two full modules to the space station and removing two whose propellant level is either very low or zero. The removed modules will be returned to Earth and refurbished; this operation will take a lapse of time equal to the time between two shuttle flights, and thus, returned modules must stay on Earth for at least one shuttle flight. During these resupply operations, modules that are not empty can be moved from a parking space to another. Considering the levels of propellant required for typical reboost operations, Level II Space Station Freedom Resource Utilization Analysis Office estimate that there will be approximately a delivery of two modules every year. Besides reboosting, propellant is needed for *attitude control*. The requirement for attitude control is estimated as a fixed quantity equally distributed among the four corners of the rectangle where modules are parked.

The space station must have enough propellant to continue operation in spite of several contingencies. An operation called *collision avoidance maneuver* requires fixed (and unequal) quantities of propellant from the four corner of the rectangle. Additionally, normal reboosting should be possible if a scheduled shuttle flight is canceled, i.e. the space station should be able to perform a *skip cycle* reboost even if the canceled shuttle flight is one in which propellant was to be supplied. The requirements for a skip cycle are those of a reboost and attitude control. It must be underlined, that these contingency propellant requirements are not actually consumed, but must be available.

3.2 Propellant resupply schedule form

A schedule for all the operations regarding propellant is needed. The schedule should answer four basic questions:

- What triangle is to be fired on each reboost operation?
- On which shuttle flights should propellant modules be supplied to the space station?
- For each propellant resupply operation, which modules should be removed, and where should the new modules be placed?

- For each propellant resupply operation, which modules, if any, should be moved? and to which corners of the rectangle?

3.3 Optimization goals

The complete criteria for defining what is a good schedule has not been fully determined yet. The following elements are known to be desired in a good schedule.

- Robustness:
  A good schedule should be capable of tolerating small changes due to modifications or cancellations in shuttle flights, variations in solar activity, requirements of schedules for the supply of other items, etc.

- Wasted propellant:
  Propellant contained in modules returned to Earth is a waste and should be minimized. A efficient schedule should ask for the return of modules only when these are empty or almost empty.

- Propellant resupply operations:
  A good schedule will require the minimal number of propellant resupply flights.

- Module movements on the space station:
  As explained before, non empty modules can be moved from one to corner to another during a resupply operation. A good schedule should make the least number of these movements.

3.4 Simulator outline

A simple simulator, written in C and running on the Macintosh and on UNIX workstations, was developed for this problem. The simulator allows a user to type in instructions regarding the use of propellant and its resupply. For each shuttle flight, the user is asked if a resupply should occur; if so, it is then asked if modules should be moved from one corner to another, which modules should be returned to Earth and where should the new modules be placed. The user is also asked which triangle should be fired on each reboost operation. The simulator displays the placement of each modules and its propellant contents, and takes into account the propellant requirements for contingencies. The propellant content of the modules is displayed to the user after each operation. In this way, a whole schedule can be tested step by step.

In summary, the user interacts with the simulator by making decisions regarding the following operations:

- Reboost:
  The user decides which triangle of propellant modules is to be fired on each reboost cycle. The simulator displays the contents of the modules. In case of a schedule failure, i.e. the user has drained modules so that a reboost is impossible, the simulator traces back in time to the last viable situation so that the user can continue simulation from this point.

- Resupply of propellant:
  The simulator allows the user to decide when to perform a resupply, which modules to return, and where to place the new modules.

- Movements of modules:
  If a resupply is to occur, the simulator asks the user which modules are to be moved, and which are the target corners. The user can enter as many movements as desired.

3.5 Simulator reinterpretation of commands

An important characteristic of the simulator, which is necessary if it is to be coupled to a genetic algorithm, is its capacity to reinterpret commands when they cannot be carried out.

When the user requests an operation that is not valid, the simulator is capable of either ignoring the request or finding the next possible operation of the same type which is possible. The following is a list of the reinterpretations the simulator performs:

- Try other triangle
  When the user attempts to perform a reboost using a triangle whose modules do not contain enough propellant, the simulator tests the other triangle, and if valid, it fires it.

- Two consecutive resupply operations
  Because of the requirement that modules that return to Earth must stay for at least one shuttle flight, resupply operations cannot be consecutive. The simulator ignores any attempts to perform consecutive resupplies.

- Removing empty modules
  During a resupply operation, empty modules must be removed. The simulator ignores attempts to remove modules that are not empty if other that are empty are not removed first.

- Removing least full module of corner
  During a resupply operation, the least full module of a corner must be removed. When specifying modules to be removed, the user only indicates the desired corner; the simulator removes the least full module from that corner.

- Placing modules on corners that are full
  Each corner of the rectangle contains two parking spaces for modules. If the users tries to place a module on a corner that already has two modules, the simulator will place it on the next corner that has an empty parking space.

These reinterpretations of user commands are important when the simulator is coupled to a genetic algorithm, because in this way many invalid individuals are expressed as valid schedules, and thus, are efficiently employed in the genetic search. In effect, through these transformations, the search space is pruned of regions that contain only invalid solutions.

4. Coupling the simulator to Splicer

There are two main issues to resolve when coupling a simulator, like the one developed for this work, and a genetic algorithm tool such as Splicer. First, a way to encode possible solutions into binary strings must be chosen. Second, a method to evaluate this solutions must be defined. In the following section these matters are explained.

4.1 Encoding of schedules into binary strings

The possible schedules must be encoded into binary strings, or chromosomes, so that the genetic algorithm can operate over a population of them. Thus, an encoding is a transformation from the space of possible schedules, i.e. lists of commands that can be given to the simulator, to the space of binary strings. In this encoding design the chromosome is composed of four segments and each segment encodes one of the possible commands to the simulator.

- Triangle selection
  This segment consists of as many bits as reboosts being scheduled. Each position indicates with a 0 that triangle A, or with a 1 that triangle B, is to be fired at the corresponding reboost.

- Is resupply to be performed?
  This segment consists of as many bits as reboosts being scheduled. Each position indicates if a resupply is to be performed.

- Selection of corners for resupply

This segment consists of as many subsegments as necessary to represent the maximum number of resupplies thought necessary for the space station to operate for the lapse of time being scheduled. Each subsegments contains eight bits in groups of two. Every two bits point to one of the four corners.

- Selection of modules to be moved
  This segment consists of as many subsegments as necessary to represent the maximum number of module movements thought necessary. Each subsegment is composed of four bits. The first two bits indicate which module is to be moved and the last two indicate the corner to where it will be moved. Notice that during a resupply, two modules must be chosen to be removed so that there are only four left to be moved to different corners.

Each chromosome, or individual in genetic algorithm, encodes the operations a user would request from the simulator. Even though it can be handled by the simulator that was developed, the selection of modules to be moved was not incorporated into the genetic search. This step remains for future work.

4.2 Evaluation

The problem of finding appropriate schedules for the resupply of propellant can be considered at two levels. First, and most important, it is necessary to find *valid* schedules, i.e. schedules that can perform all the reboosts asked for. Then, minimization of wasted propellant and maximization of robustness can be considered. In a genetic algorithm, where initial individuals are generated randomly, it is very unlikely that valid solutions will be contained in the initial population. Thus, the first goal of the evaluation procedure should be to drive the population to valid solutions. In light of the above, the objective function chosen had the following form:

$$\text{fitness} = \text{number of successful flights} \cdot 10 \\ - \text{total wasted propellant} / 1000.$$

The maximization of robustness has yet to be considered. The following section describes ways under consideration to include robustness in the optimization.

5. Maximization of robustness and multi-objective optimization

The objective of maximizing robustness in a schedule requires a different approach than that used to attack all other optimization goals. Robustness is difficult to measure. In the context of resupply to the space station, a robust schedule is one that can tolerate small changes due to modification or cancellations in shuttle flights, variations expected in solar activity, requirements of schedules for the supply of other items, etc. What is meant by "tolerate small changes" is yet to be defined. Additionally, robustness is a characteristic that will surely conflict with other optimization goals; for example, a schedule that requires for propellant resupply more often than necessary will probably be more tolerant to shuttle flight cancellations, but will waste propellant.

The problem of maximizing robustness can be addressed in two steps. First, develop a stochastic method to measure robustness of a schedule and incorporate it into the objective function. Second, the problem will be treated as one of multi-objective optimization where robustness will be maximized independent by of all other objectives. In the following subsections the proposing method for applying the genetic algorithm is presented.

5.1 Measuring robustness

Genetic algorithms are know to be tolerant to noise in the evaluation of the objective function (Fitpatrick, Grefenstette, and Van Gucht, 1984). The reason for this is that the genetic algorithm implicitly processes *schemata* which are patterns of bits that are represented by many members of the population. In this way, the evaluation of a signal individual can be corrupted with noise, and the genetic algorithm will continue to find near optimal solutions, as long as the noise has zero mean. Therefore, for each individual schedule in the population, a number of random small changes will be generated. The robustness will be obtained as the average performance of the modified schedules; in the expected value, this will be a correct measure of robustness. This value of robustness can be incorporated into the objective function as part of a linear combination of all the optimization goals. In this way, for a given relation of relative importance among the optimization goals, the genetic algorithm will find a near optimal solution.

5.2 Multi-objective optimization with genetic algorithms

Many real optimization problems, like the scheduling of supplies to the space station, require that several criteria be optimized simultaneously. In these problems, some of the objectives are conflicting, and it is difficult to decide the relative importance of each one. In practice, all the criteria are usually combined into a single objective function by taking a linear combination of them, in order to avoid the problem of multi-objective optimization. This is not always the best approach, but one often taken because of limitations on the optimization methods.

The area of multi-objective optimization has not been fully attacked by genetic algorithm researchers. Schaffer (1985) has presented a method for applying genetic algorithms to this type of problem, and no continuation of this effort has been reported. Goldberg (1989, p. 201) suggested applying a combination of rank selection (Baker, 1985) and niche and speciation methods (Deb and Goldberg, 1989). An extension to the previous efforts, (Valenzuela Rendón & Cantú Aguillen, 1992) that apply niche and speciation methods to the solution of mutlimodal problems to include multi-objective optimization as proposed by Goldberg, is proposed to attack this problem. In this way, the genetic algorithm will find not one but a family of near optimal schedules for varying degrees of relative importance of robustness versus all other optimization goals.

6. Final Comments

The Space Station Freedom will require the resupply of many items over a long period of time. Schedules for individual items will first be designed, and thus, these schedules will be integrated into a single schedule. For this integration to take place, individual schedules must be able to tolerate small changes. Genetic algorithms seem like an appropriate tool to apply to this problem in view of their proven ability to solve complex objective functions and their robustness as search methods.

As a first step, we have developed a simple simulator and an encoding by which a genetic algorithm can be applied to finding schedules for the resupply and use of propellant without considering robustness. As following steps, we will take advantage of the ability of genetic algorithms to tolerate noise in the objective function, and measure robustness in a stochastic manner. Also, we propose to apply multi-objective optimization techniques in genetic algorithms to find families of near optimal schedules for varying degrees of relative importance of robustness.

Acknowledgments

This work was performed during a six month stay of Manuel Valenzuela-Rendón at NASA's Software Technology Branch as part of RICIS (Research Institute for Computer and Information Systems) research activity number SR.04 (NASA Cooperative Agreement NCC-9-16) and with partial support of the ITESM.

References

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 101–111.

Etter, D. M., Hicks, M. J., & Cho, K. H. (1982). Adaptive genetic algorithm for determining optimum filter coefficients in recursive adaptive filter. In *IEEE International Conference on Acoustics, Speech and Signal Processing,* Vol. 2, pp. 635–638.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. *Proceedings of an International Conference on Genetic Algorithms and Their Applications,* pp. 162–164.

Deb, K., & Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. *Proceedings of the Third International Conference on Genetic Algorithms,* pp 42–50.

Fitpatrick, J. M., Grefenstette, J. J., & Van Gucht, D. (1984). Image registration by genetic search. *Proceedings of IEEE Southeast Conference,* 460–464.

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International, 44*(10), 3174B. (University Microfilms No. 8402282).

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Goldberg, D. E. & Lingle, R. (1985). Alleles, loci, and the traveling salesman. *Proceedings of an International Conference on Genetic Algorithms and Their Applications,* pp. 154–159.

González Sáenz, A. (1991). *Algoritmos genéticos: Una aplicación al diseño de filtros digitales* [Genetic algorithms: An application to digital filter design].Unpublished master's thesis, ITESM, Monterrey.

Guerra-Salcedo, C. M., & Valenzuela-Rendón, M. (1991). Resolviendo consultas de apareamiento parcial utilizando algoritmos genéticos [Solving partial match queries by means of genetic algorithms]. *Memorias de la VIII Reunión Nacional de Inteligencia Artificial,* 13–28. Sociedad Mexicana de Inteligencia Artificial.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Shaffer, J. D. (1985). Multi-objective optimization with vector evaluated genetic algorithms. *Proceedings of an International Conference on Genetic Algorithms and Their Applications,* pp. 93–100.

Valenzuela-Rendón, M., Guerra-Salcedo, C. M., & Icaza, J. I. (1991). A genetic algorithm approach to partial match retrieval based on hash functions. *Proceedings of the IV International Symposium on Artificial Intelligence,* 156–162.

Valenzuela Rendón, M. & Cantú Aguillen, C. (1992). *Solución a sistemas de equaciones no lineales simultáneas usando algoritmos genéticos.* [Solution to nonlinear simultaneous equations by genetic algorithms]. Manuscript submitted for publication.

Wang L. (1991). Genetic Algorithm Overview, *proceedings of The 1991 Science, Engineering & Technology Seminars*