

Computer Science Masters Project
Genetic Algorithms for Autonomous Satellite Control

By

Thomas R. Kiehl

April 1999

Approved by:

Project Advisor: Dr. Piero Bonissone

Signed: _____ Date: / /1999

Sponsoring Faculty Member: Dr. Michael Skolnick

Signed: _____ Date: / /1999

Genetic Algorithms for Autonomous Satellite Control

Thomas R. Kiehl
K1-5C25
GE Corporate Research
P.O. Box 8
Schenectady, NY 12301

Abstract

Genetic Algorithms have been applied successfully to many problems in a variety of different domains. Many current applications deal with various scheduling problems. These scheduling problems range from job shop scheduling to the travelling salesman problem. This paper will show how genetic algorithms can be used to schedule anomalous tasks on military and commercial satellites, which have as their objective to avoid unnecessary downtime. The tasks, which are scheduled on the satellites, have various types of constraints associated with them. It will also be shown how these constraints are handled at various stages of the genetic algorithm. Several crossover operators will be tested and the final results will be compared with results generated by a linear programming implementation of the problem.

1. Introduction

Genetic Algorithms have been successfully applied to many problems characterized by high numbers of nonlinear variables. Among these applications are a whole host of problems which can be categorized as scheduling problems. Many of these problems are directly related to the

travelling sales man problem. While this problem is not directly related to the traveling salesman problem it is a scheduling problem.

The object of our problem is a constellation of satellites in Low Earth Orbit. These satellites' orbits do not follow a regular pattern. Thus the schedule for any one of the 25 satellites doesn't necessarily follow the same orbit twice within a finite period of time. As these satellites follow their orbits they provide services to the cities which they come into contact with.

A given city on the surface sees a steady stream of satellites appearing on the horizon, remaining in view for a period of time (most likely different from the last time) and then disappearing over the horizon. A city may have several satellites in view during any given period of time. These satellites provide a variety of services to the cities while they are in view.

The company which owns the satellites is responsible for providing a certain level of service to customers in these cities of interest. While the satellites cannot be re-allocated, that is, they cannot be moved into another orbit in order to provide more service to a city which is not in it's immediate path. Thus, it is to the company's advantage financially to ensure that these cities are receiving the highest level of service possible.

Often a city is even provided with more service than is necessary. However, the goal in our project was to maximize overall coverage.

It is our job to schedule anomalous tasks on these satellites. These tasks must be scheduled in such a manner that they do

not infringe on the satellite's ability to provide services.

This paper will explain the representation used, a comparison between a few different types of crossover. Also, several methods for handling constraints at different points in the execution of the genetic algorithm. Finally results will be compared to those attained through a linear programming method.

2. Problem Formulation

There are many pieces of information which go into the formulation of a solution. These include the orbit data for each satellite in the constellation, the set of tasks which must be scheduled, and any interactions between the tasks.

Tools exist which predict each satellite will be for a period of time in the future. Given this information along with knowledge of where cities and other points of interest exist we can determine when a satellite must be available for providing services to various cities.

The set of tasks to be scheduled is determined dynamically by human agents who monitor the state of the satellite constellation. The interactions between these tasks and other activities on the satellite have been predetermined.

2.1 Coverage

When over a city or other point of interest, the satellite will provide services to that entity. Scheduling maintenance tasks on a satellite while it is providing services to a city will decrease the effectiveness of the satellite. It is preferred that these tasks be scheduled during periods of time when the satellite is not providing services. For example we may not want to schedule a maintenance task immediately if the satellite is active (or providing services) but if we similarly delay a period of time, we can then successfully schedule the task without infringing on this active time period.

A satellite's predetermined schedule of locations consists of a set of events, each of these pertaining to a particular period during which the satellite is available to provide services to a city. These periods are often referred to as coverages. A coverage is described in terms of a location or city name, the time at which the coverage begins, and the time at which the coverage ends. Table 1.1 contains an excerpt from a typical schedule for a single satellite.

Table 2.1

Sat.	City	Instance	Start	End	Priority
1	Beijing	1	27.01	55.06	2
1	Beijing	2	157.86	186.3	2
1	Beijing	3	289.76	311.11	3
1	Beijing	4	878.41	893.21	3
1	Beijing	5	999.66	1027.08	2
1	Beijing	6	1129.25	1157.81	2
1	Beijing	7	1263.34	1289.86	3
1	Beijing	8	1397.66	1424.56	2
1	Bombay	1	159.56	173.91	3
1	Bombay	2	287.46	314.55	2
1	Bombay	3	417.51	445.26	1
1	Bombay	4	555.91	559.96	2
1	Bombay	5	1116.15	1143.41	2
1	Bombay	6	1246.2	1273.75	2
1	Bombay	7	1385.6	1401.85	3
1	BuenosAires	1	95.01	123.75	3
1	BuenosAires	2	226.01	251.86	2
1	BuenosAires	3	934.91	960.91	2
1	BuenosAires	4	1063.16	1091.86	3
1	BuenosAires	5	1197.4	1223.05	3
1	BuenosAires	6	1356.76	1357.93	3
1	Calcutta	1	28.31	40.01	3
1	Calcutta	2	159.05	183.15	2
1	Calcutta	3	288.51	317.25	2
1	Calcutta	4	420.8	443.61	2
1	Calcutta	5	993.5	1014.65	2
1	Calcutta	6	1118.9	1147.56	2
1	Calcutta	7	1252.16	1277.16	2
1	Calcutta	8	1394.61	1407.4	3

The times when a specific satellite is covering a specific city do not form a regular pattern and the times between different coverages varies widely. Tasks must be scheduled in the periods of time between these coverages lest a non-optimal schedule may be produced. An ideal schedule would cause no shrinkage of coverages.

The schedule of coverages is provided as input to our Genetic Algorithm. This is generated by a satellite simulator and

formatted in such a way that the GA can use it in evaluating the fitness of an individual.

2.2 Tasks

There are a variety of possible maintenance tasks which could be scheduled during a period of time. Each of these tasks have certain restrictions and rules which they must be executed with respect to. The tasks include...

- **Battery Offline** : Executed when a battery must be put offline
- **Battery Recondition** : Forces a full reconditioning of a battery. This event must be scheduled to occur on a somewhat periodic basis. Batteries require full reconditioning after spending a period of time being intermittently charged and discharged
- **Momentum Dump** : A fairly short task which allows a satellite to maintain it's orbit
- **Keep Station** : this task is also related to the mechanics of maintaining orbit
- **Verify Link** : This test the satellite's ability to communicate with ground stations.
- **Calibrate** : General maintenance
- **State of Health** : send a measure of the satellite's state to a ground station

2.3 Events

The various tasks have been classified into four types of tasks. Each of these types has specific characteristics of interaction with other events as well as other tasks. Before describing the classifications of tasks, the various types of events, which a task may interact with, must be described. There are five different types of events which a task could possibly interact with depending on when it is scheduled relative to whatever other events are occurring on a given satellite.

- **City**: This type of event is a period of coverage over a city. These events correspond to periods of time when the

satellite is close enough to a city to provide services to it.

- **Ground Station**: This is a period of coverage over a Ground Station. These events indicate periods of time when a satellite is near enough to a ground station to communicate with it.
- **Eclipse**: This kind of event indicates any period of time when the satellite is not in sunlight
- **Task of Type 1,2,3**: This indicates any period when a satellite is executing a task of type 1, 2 or 3.
- **Task of Type 4**: This indicates any period when a satellite is executing a task of type 4

2.4 Interactions between Tasks and Events

The interaction between tasks and previously scheduled events, on a given satellite, can be characterized in one four possible ways. Firstly, a task can cause an event to "shrink." This type of interaction indicates that if a task is scheduled at the same time as an event of a given type, the event will shrink. For example a task of type 1 shrinks city coverage events. Thus if a task of type 1 were scheduled on a satellite starting at $T=200.13$ and ending at $T=220.13$ and that satellite had a city coverage scheduled from $T_s=210.43$ (start) to $T_e=335.14$ (end) then the city coverage would shrink to accommodate the task. After the sample task is scheduled, the city coverage would now begin at $T_s=220.13$. This type of relationship is indicated by the word "shrinks" in table 1.2.1.

A task could also have no interaction with events of a given type. If this is the case a task can be scheduled at any time and would have no effect on events with which it had no interaction. This relationship is indicated by "No Interaction" in table 1.2.1.

The last two interactions impose certain constraints on when a task can be schedules with respect to events. First, a task may not be allowed to overlap with a given event. In this case if a task were scheduled

such that it overlapped with an event which it was not allowed to overlap, this would result in an invalid schedule. This constraint is represented by “No Overlap” in table 1.2.1. One example of this relationship is the relation between tasks of type 2 and eclipses. A satellite’s schedule would be invalid if a task of type 2 was scheduled at a time when the satellite was in eclipse

Table 2.4.1

Type	City	Ground Station	Eclipse	Interaction with types 1,2,3	Interaction with type 4
1	Shrinks	Shrinks	No Interaction	No Overlap	No Interaction
2	Shrinks	No Interaction	No Overlap	No Overlap	No Interaction
3	No Interaction	Within	No Interaction	No Overlap	No Interaction
4	No Interaction	Within	No Interaction	No Interaction	No Overlap

The last type of interaction restricts the time when a task can be executed by stating that the task must be executed at the same time that an event of a particular type is also occurring. One example, taken from table 1.2.1, indicates that tasks of type 4 must occur during a period of time when the satellite can maintain contact with a ground station. Here is a sample set of tasks which might be scheduled on a single satellite during a twenty four hour period.

Table 2.4.2

param	Tasks	dur	start	after	finish	by :=
1	BatteryRecond	1	GS_Any	50	0	1440
1	Calibrate	1	GS_Any	0.34	0	1440
1	KeepStation	1	GS_Any	16	0	1440
1	StateOfHealth	1	GS_Any	10	0	720
1	StateOfHealth	2	GS_Any	10	360	1080
1	StateOfHealth	3	GS_Any	10	720	1440

3. The Genetic Algorithm

Genetic Algorithms are capable of traversing highly nonlinear fitness terrains and capitalizing on information gleaned from complex relationships. Our initial use of the genetic algorithm could possibly be

viewed as overkill since, as of this first stage of the project, there are no interdependencies between the schedules of the various satellites. The entire constellation of satellites could be optimized simply by optimizing each individual satellite’s schedule. Future constraints that will be placed on the system indicate that this will not always be the case.

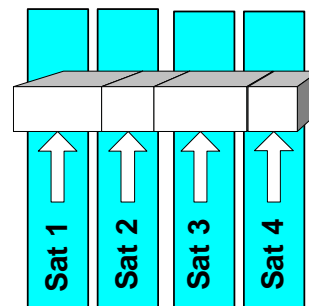
In the current phase however there are interactions between the tasks that must be scheduled on a particular satellite. Also, the fitness of a population is cumulative fitness of all of the individuals in the population.

3.1 Representation

The representation used in this implementation takes advantage of some of the key features of the tasks that we are scheduling. Given a set of tasks, which must be scheduled on the constellations of satellites, we know that they must be one of a set of classified tasks. Each type of task has a fixed duration. The only information that is necessary in the chromosome, is the starting time of a given task. The set of tasks is then ordered and each task is associated with a particular position on the chromosome. Tasks that are to be executed on the same satellite are found together on the chromosome as pictured in Figure 3.1.1.

The alleles of the chromosome are real values. Each real value representing the

Figure 3.1.1



number of seconds which are to elapse

between the beginning of the simulation period the time at which the task associated with that allele should begin.

Other data structures hold the rest of the information about the tasks. This data is then used by the fitness function to determine a measurement of the feasibility of the set of start times.

3.2 Fitness Function

The fitness of an individual is essentially a measure of how much coverage the constellation of satellites is providing during the period of time being scheduled. The fitness value is directly based on how well the individual preserves the original schedule of coverages. The maximum fitness a particular individual can possibly receive is equivalent to the total amount of coverage provided by the original schedule prior to scheduling any anomalous tasks. If the total coverage obtained by an individual were equal to the total coverage possible we would be assured that no shrinkage of the available coverages had occurred and that we had reached an optimal schedule.

If the original schedule were to simply contain the coverages displayed in table 1.1 the maximum fitness would be the sum of the durations of each of the coverage events.

To compute the fitness of an individual, the fitness function uses the information in the chromosome to determine when the tasks take place in the schedule of the satellites. These tasks are superimposed on the schedule of events, the events are shrunk appropriately and the final total coverage is computed.

The Fitness function employed in this project also incorporates a penalty function to handle some specific types of interactions between tasks. The design of the penalty function will be described in a later section.

To compute the fitness of an individual, for each satellite a data structure is constructed containing all of the coverages that each satellite provides during the simulation period. The tasks being

scheduled are then superimposed on that data structure based on the start times provided by the chromosome being evaluated. If a task overlaps with a shrinkable coverage, then that coverage is truncated to reflect the overlap. Once all of the tasks have been superimposed, the total remaining coverages are summed. The remaining coverages on each satellite are then summed with each other to provide a fitness value for each individual.

In the event that two incompatible tasks are scheduled on a single satellite such that they overlap, a penalty is induced. Incompatible task types are contained in table 2.4.1. The penalty function will be outlined in greater detail in section 4.0.

3.3 Steady State GA

A Steady State GA was employed which maintained a certain number of the best individuals in one population into the subsequent generation. About half of one population was maintained through the next generation.

This was tested against a full replacement GA. On average the Steady State GA was able to progress to a solution much quicker than the full replacement method.

3.4 Mutation

Two different mutation operators were tested in various scenarios. One of which was fairly “catastrophic.” This method caused a great amount of change in the chromosome of the individual being mutated. A new valid number was generated for each allele in the chromosome. This was deemed appropriate being that we were dealing with real number alleles. In many cases simply changing the values by selecting new numbers which were normally distributed around the actual values did not increase the diversity in the population to enable large jumps across the search space. It was determined that selecting numbers in a normal distribution around current allele

values only served to find neighboring positions in the search space, not new space entirely, which is typically desired in a mutation operator. These tests will be explained in further detail in a later section.

3.5 Crossover

Three different types of crossover were employed at different periods of this project. These will be contrasted in detail in a later section of this paper.

The simplest form of crossover which was employed, was a single point crossover method. This was quickly written off as ineffective. This was mainly due to the inability of this type of crossover to allow the individuals to explore beyond the space allowed by the initial. The GA was unable to make local fine tuning adjustments due to the lack of the necessary genetic material.

Another crossover method which was employed was Blx crossover. This too was not terribly effective in comparison to the final method that was chosen. The Blx method was more likely to generate individuals which were not at all similar to either parents, rather than the children being somewhat similar to each parent. The exact implementation and possible enhancements to our implementation will be discussed in depth later in the paper.

The final, and most effective, crossover method employed was devised to imitate the results of single point crossover when applied to binary valued alleles. It was observed that when single point crossover is applied to two binary strings, components of the children distinctly resemble each of the parents. Crossover by Parent-Based Distributions generated real valued children which distinctly resembled their parents while assuring that our genetic material is not restricted by the actual discrete values which resided in our initial population. The genetic material in our population is however controlled such that the material available to us contains all of the points which are normally distributed around the values in the initial population.

Again, this method will be described in greater detail below.

4.0 Handling Constraints

There are several types of constraints which must be handled by this application. Constraints can be handled at various points in the execution of the GA. Each type of constraint will be described in detail and then the method for handling that constraint will be detailed as well.

These constraints can be classified into two categories. Firstly, *Static Constraints*, which are predetermined before the GA is executed. One such case is the constraint placed on the reconditioning of batteries. This type of task cannot in any way overlap with an eclipse. Since we know this ahead of time, we can enforce certain constraints to ensure that these two do not overlap.

The second category, *Dynamic Constraints*, are not known ahead of time and must be handled in a very different manner. One example of a dynamic constraint is the restriction that certain tasks cannot overlap with other tasks. Since the tasks are scheduled dynamically we cannot use the same types of constraints which we

$$Sim_{start} \leq Task_{start} \leq (Sim_{end} - Task_{dur})$$

applied to the static problem

4.1 Static Constraints

A static constraint is a constraint which does not change during the entire running time of the GA. Looking back at table 2.X.X we can see three different cases where a static constraint can be enforced.

The first case is the interaction between tasks of type 2 and eclipse events. We know that these types of tasks cannot ever overlap an eclipse. What makes this a static constraint is the fact that we know, prior to running the GA, when the eclipse is

going to happen. Therefore, we can plan ahead to be sure that we meet this constraint.

The second and third static constraints displayed in Table 2.X.X are a similar type of constraint between tasks of type 3 or 4 and ground stations. When these tasks are executed the satellite must establish a communication link with a ground station and then execute the task while that link is maintained. Thus if a particular ground station is in view for four minutes and the task will take five minutes to execute, then the task cannot be executed in conjunction with that ground station. Tasks of type 3 or 4 must maintain contact with a single ground station. The satellite cannot begin it's task in conjunction with one ground station and then complete it in conjunction with another ground station.

4.1.1 Restricting Start Times

Both of the cases described above can be handled by limiting the valid times at which a particular task can start. Our GA already limits task starting times. The GA ensures that all tasks will start after the start of the simulation, and end at or before the end of the simulation. Therefore, the base restriction on start times restricts a task's starting time (T_{start}) to some time between the beginning of the simulation (Sim_{start}) the end of the simulation period (Sim_{end}) less the duration of the task (T_{dur}).

This restriction is then placed on the

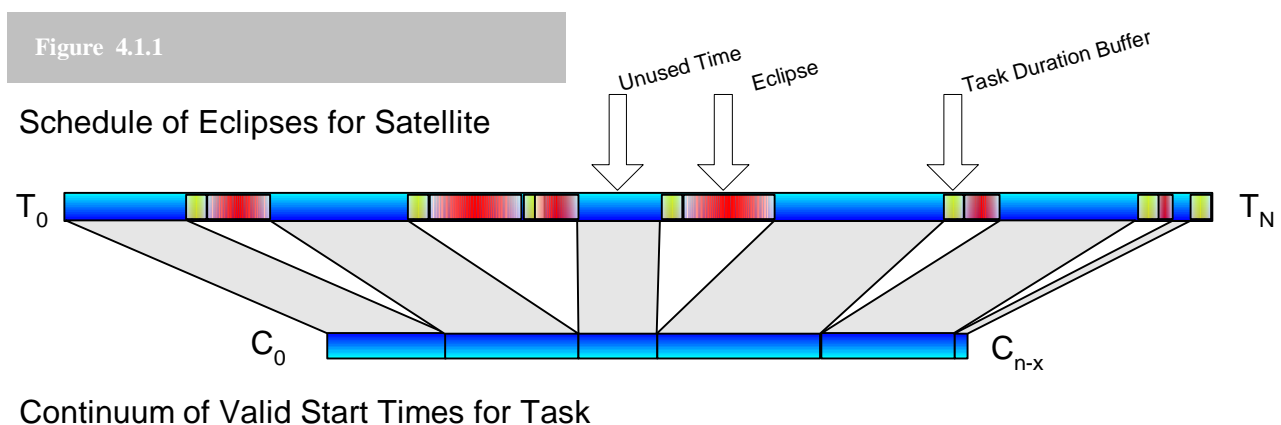
allele value that corresponds to any given task. With this restriction in place we are assured that any value which that allele holds will remain in the valid range. This restriction is referenced any time an individual is initialized and when the individual is mutated and even when a new individual is generated through crossover. When a new individual is generated, the initialization routine will reference the constraints for each allele value in the individual and will generate only values in the given ranges.

In the same we want to use these ranges to restrict the starting times of certain tasks so that we exclude the possibility of that task ever being scheduled such that it collides with a static event. First we will look at using these constraints in the case of the relationship between tasks of type 2 and eclipse events.

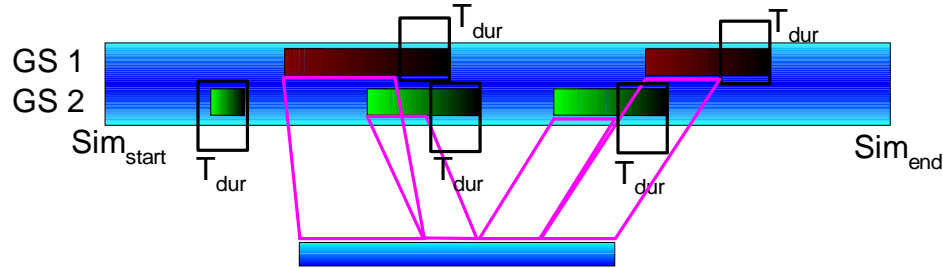
4.1.1.1 Handling Eclipses

Prior to running the GA, we know exactly when each satellite will be in eclipse. We also know if and how many tasks of type two are to be scheduled on each satellite during the duration of the simulation. We are also aware of which allele in the chromosome corresponds to each of these tasks. It is our goal to restrict the range of valid values for these alleles such that any value in one of those positions will correspond with a valid starting time for that task.

To calculate the range of valid



Ground Stations Visible to a Single Satellite



Range of Valid Start Times for Task

starting times for a task which is restricted by eclipse we must look at the schedule of eclipses for the satellite which the task intends to run on. We can potentially run our task within any gap, of sufficient size, between eclipses. If a given gap is larger than the duration of the task, then we can include a certain portion of that gap in our valid range of start times. If

$E_{i_{start}} - E_{i-1_{end}} > T_{dur}$ then we can include a portion of that range in our final range of valid start times. Within a gap, of sufficient size, only a certain range of values are actually valid start times. Valid start times within a range can be found by subtracting the task's duration from the gap's full range. Thus the range of valid start times contained within a gap corresponds to $E_{i_{start}} - E_{i-1_{end}} - T_{dur}$. This calculation must be carried out for every gap on a satellite. Then finally, the results can be summed to give us an actual range of valid starting times.

This range is then applied to the appropriate alleles and we are then assured that these tasks will not collide with eclipses. This does add some complexity to the calculation of final fitness. Every time we need to determine when this particular task actually begins, we have to then convert the actual allele value into a real starting time.

Figure 4.1.1 demonstrates the aforementioned mapping graphically. Notice that there is a space between the second and third eclipses, which is not long enough to execute the task. Therefore, that

space has not been mapped into the range of valid start times. Thus, when an allele represents this task, the number stored at that allele must be in a range of values on the C timeline. In the execution of the GA, the fitness function must then map that number onto the T timeline to determine if the task impacts the fitness of the individual.

4.1.1.2 Handling Ground Station Interaction

The required interaction with ground stations are handled in much the same way that eclipses are handled. There is a similar mapping which takes place when dealing with tasks which are required to execute in view of a ground station.

The significant difference between handling ground stations and handling eclipses is that when scheduling around eclipses we try to schedule in the period between eclipses. Yet, for tasks which must occur while the ground station is nearby, we try to schedule the task between the beginning and end of the exposure to a single Ground station.

Another significant difference between the ground station constraint and the eclipse constraint is the fact that multiple ground stations can be visible at any one time. Figure 4.1.2 shows how when we map from the actual timeline to the interpreted time line, we must sequentialize each of the exposures to the ground stations. If we choose a value as our start time, in the shortened range of valid times, this will also determine which ground station we must

execute the task in conjunction with. Also notable from Figure 4.1.2 is the fact that we can have an exposure to a ground station which is too short for us to fully execute the task at hand. Thus, this exposure is not factored into the range of valid start times.

In both the case of the eclipses and the case of the ground stations, we handle these specifications by enforcing the constraints on specific allele positions. Thus this is less complexity which must be dealt with by the actual evolutionary mechanism of the GA. While it does add some computation time to our fitness function, this is significantly less time than would be required if we were to withhold this information from the GA.

4.2 Handling Dynamic Constraints

Dynamic constraints are much more complex than static constraints. A dynamic constraint in essence is an interaction between different allele positions on the same chromosome. Dynamic constraint handling is necessary in this project is due to the interactions of specific types of tasks.

Going back to table 2.x.x we can see that tasks of type 1,2 or 3 cannot overlap with other tasks of type 1,2 or 3. Also, tasks of type 4 cannot overlap with other tasks of type 4. This imposes a dynamic constraint because we don't know when all of the tasks are going to be scheduled until an individual is generated. Also, one individual will schedule each of the tasks at different times than others. Hence, we cannot enforce a static constraint as in the previous problems. This constraints must be handled at run time.

Our implementation uses a simple penalty function based on the number of collisions which occur in an individual's evolved schedule. An inflection point was predefined and utilized in the function. Whenever the number of collisions was beyond that point, the individual received very small fitness values. Whenever the number of collisions was less than this point then the fitness was also affected by the number of collisions, but not as drastically. The fitness was determined by equation

4.2.1. In this equation I is the inflection point, C is the number of collisions in the individual's schedule, and F_{pre} is a preliminary accounting of the individual's fitness without the collisions factored in. Thus, if an individual's schedule evidenced no collisions, then it's final fitness would be equivalent to it's preliminary fitness.

Equation 4.2.1

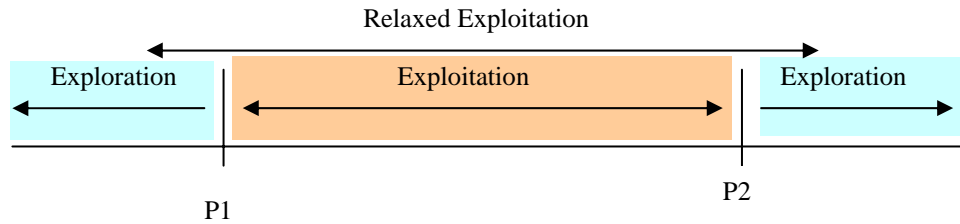
$$F = \begin{cases} F_{pre} - ((F_{pre} / I) * C) & C \leq i \\ 0.0001 & C > i \end{cases}$$

5.0 Comparison of Crossover Methods

Three different crossover methods were employed, each with very different results. Performing crossover on real valued chromosomes is a problem to which many different approaches have been applied. One of the methods, single point crossover, is typically applied to binary representations. We used this method simply as a baseline for performance. The second method was loosely based on the Blx method. This method was not fully implemented or explored. The final method that was applied was designed to be similar in effect to the single point crossover method, yet tuned to a real numbered genome.

5.1 Uniform/Single Point Crossover

The very common Uniform crossover method builds children by randomly choosing allele values from one parent or the other. Thus a child would contain some number of allele values from one parent and the rest of its allele values would be from the other parent, while the other child is simply the opposite. This method is typically employed for integer valued genomes. It is rarely useful for real genomes particularly if the fitness function is very sensitive to minute differences in the allele values. Uniform crossover will be employed as a comparison point but it is not expected to perform well for the aforementioned reasons. Uniform crossover



typically isn't effective for real valued genotypes.

Single point crossover is another method which is employed quite often in conjunction with integer valued representations. To generate two new children from two parents using single point crossover, a random point in the chromosome is chosen. One child will then be composed of the segment before that point from one parent and the segment after that point from the other parent. The other child would be the reverse.

Again, this method is typically employed with integer based representations and works remarkably well in those cases. This method will not be tested as it is simply a variation on the Uniform crossover method which is not expected to fair well.

5.2 Blx Crossover

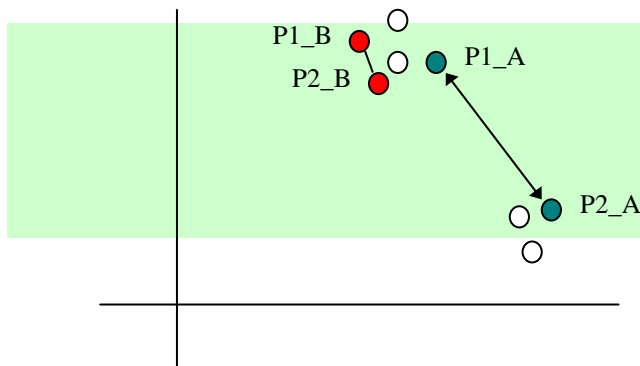
This method, which was proposed by Herrera, Lozano & Verdegay [3], is much more applicable to real numbered genotypes. This method divides up the space around the parents and labels these areas as being areas of exploitation,

exploration or relaxed Exploration. The way the algorithm is defined, two parents generate four offspring, and only the best two are passed to the next generation. Figure 5.2.1 shows how the search space can be characterized with respect to the parents. The four children are generated in different parts of the space. Typically one child is generated in the right hand area of exploration, and one in the lower while another child is generated in the area of exploitation and the final individual is generated in the area of relaxed exploitation.

Each individual is generated such that it's values are uniformly selected from the particular area determined for that individual. The side effect of this is that each child could potentially contain totally new genetic material. While this is beneficial and necessary for real numbered representations, this could easily degrade to random search.

Not mentioned in the definition of the method is any importance of the scale of the above diagram. If each of the parents are different enough from each other, the scale of Figure 5.2.1 could define each area

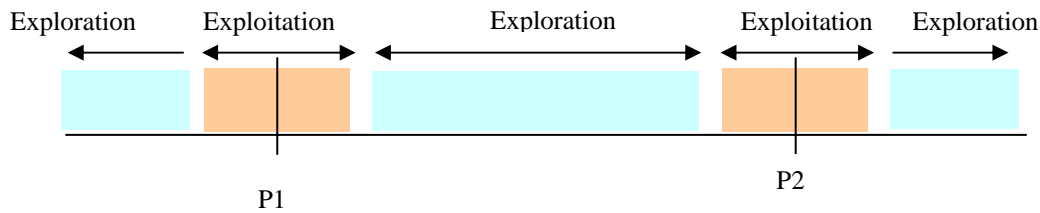
Figure 5.2.2



Assume that P1_A and P2_A are a large distance apart, while P1_B and P2_B are quite close to one another.

Choosing a point between P1_A and P2_A might be viewed as being more exploratory and less as exploitation.

Meanwhile, a point chosen between P1_B and P2_B could be viewed as a great amount of exploitation and very little exploration.



as exploration. In the event that the two parents are very close to each other it is easy to justify the classifications made as far as which areas are exploration and which are not. However, if the parents are significantly different, the space between them could be just as much exploration as exploitation. Also, if the two parents can be found at two opposite ends of the spectrum, the space currently labeled as exploitation becomes very small and thus could be viewed as more exploitation than exploration. Figure 5.2.2 explains this discrepancy visually.

Our implementation of this method was not complete. Each of the two children were generated in the “Relaxed Exploitation” space. There was also some probability that the children would just take a value directly from a parent. We felt that while this is not exactly the same as the blx method, that we had approximated it feasibly.

While this method is expected to be effective, it is not expected to be as effective as the next method which will be described.

5.3 Crossover by Parent Weighted Distributions

The final method that will be employed can be viewed as a Uniform crossover operator for real-valued genotypes. Essentially when two parents cross over, at each allele position, two new values are generated. One of these values is chosen from a normal distribution around one parent and the other value is chosen from a normal distribution around the other parent. At random, one of these values is passed to one child and the other value passed to the other child.

Comparing this method to uniform crossover as applied to binary valued

genomes we see some similarities. Using a binary representation, if we utilize an allele value from a parent we are assured that we are in a mode of exploitation. The problem with real numbered representations is making the distinction between exploration and exploitation. Again, if we utilize an actual allele value from a real numbered parent, we are positive that we are exploiting that value. However, a real valued allele can potentially have many more values than a binary valued allele, which only has two possible values. So using a value directly from a parent could be an extreme form of exploitation. Determining the range of values which to consider exploitation vs. exploration is a difficult task. This method allows you to define areas exploration as being distributions centered at each parent. This also presents another parameter by which to tune the GA. One could foresee tuning the number of standard deviations which are used to generate new values, thus widening or narrowing the area labeled as exploitation. This may be important in some problems which are very sensitive to small changes in allele values.

In implementing this method it is our goal to replicate the results which uniform crossover attains for binary representation.

6.0 Challenges and Experiments

So far we have described the problem, and explained some of the details of how we intend to apply the GA. These specifics have each be in regard to specific challenges that are expected to arise in the course of implementing this solution.

Some of the challenges will also require structured experimentation in order to fine tune the mechanics described above.

6.1 Effective Crossover

Simply the fact that we are using a real representation presents challenges. The main challenge when dealing with real numbered genotypes is the creation of a useful crossover mechanism.

As discussed earlier (section 5.0) there are several options available to us for implementing crossover. The challenge will be to choose an effective means for this specific problem. This also is a fairly straightforward set of experiments. Since each of these methods are fairly basic they can all be implemented and then tested against each other.

A series of tests will be run in which everything will be held constant except for the means of crossover. Each crossover mechanism will be run on several different random seeds. The results from each crossover method will be combined into an aggregate, then the average convergence rates of each crossover method will be compared against the others.

Tuning of the parent weighted distribution method will not be a part of the experimentation performed. This method will always use a single standard deviation to generate new values. Section 5.3 explains the importance of this distribution.

Some brief reasoning will also be presented in an effort to determine why each method performed as it did. It is expected that the crossover by parent weighted distributions will outperform the other methods for reasons cited in section 5.3.

6.2 Tuning the Penalty Function

The penalty function plays an important role in this GA. The maintenance of an entire set of constraints depends on the operation of the penalty function. The maintenance of diversity in the population is one of the main challenges that we face in tuning the penalty function.

Via the penalty function, we are applying pressure to the population to squeeze out individuals that have schedules with collisions. If we apply too much

pressure, individuals with collisions will be removed quickly from the gene pool regardless of whether or not other genetic material in that individual is valuable.

Therefore, a series of experiments will be conducted to determine the proper weight to assign to collisions. This will involve varying the inflection point in equation 4.2.1. It is expected that setting the inflection point very high will put very little pressure on the population to remove individuals with any collisions. Inversely, setting the inflection point very low puts significant pressure on the population to remove individuals with collisions as quickly as possible.

What we would like to see at the end is that individuals with collisions are still notably present in the population for a large percentage of the total number of generations. Notably present meaning that individuals would be seen at the level of most fit individual in a given population. Allowing these individuals to remain in the population creates a certain evolutionary pressure encouraging better individuals to evolve. It is surmised that by pressuring out collisions very forcefully removes a certain amount of competition. Competition, of course, being the main driver in evolutionary systems. Without this competition we are left with a difficult fine tuning problem. These experiments may even result in removing the penalty entirely.

6.3 Fine Tuning

Once we have reached a point in the evolution of a solution we may find that much of the pressure that we so heavily relied on in early generations is no longer effective or even available.

One case of this being the pressure applied against collisions. Once the population has progressed to a point where their fitness is at a fairly high level, it may be inferred that at that level, those individuals with collisions (and potentially significant genetic diversity) may not be able to survive in the population. Once these individuals have dropped out, we are

now left with a population that is fairly homogenous and complacent. These homogenous individuals will not be able to provide each other with the necessary pressure to quickly move the population to another higher fitness level. Thus we are doomed to very slowly increase our fitness relying solely on minor perturbations.

Another action that might prove to be helpful in later generations would be to cut the population size after a given duration. It is assumed that a large population is not necessary once the population is fairly homogenous. Given this assumption it will allow us to iterate over many more generations faster since each generation will be smaller. At the same time we are left in a quandary, because at the same time that we are taking this action due to a lack of diversity, we are also severely limiting the diversity.

It is end goal of the other experiments listed here to minimize the extent to which fine-tuning is a problem. Various population sizes will be tested to determine which size presents the best balance of redundant work and parallel fine-tuning.

6.5 Useful Mutation

In the case of this problem, implementing a useful mutation mechanism is also a concern. This is primarily due to the fact that many of the crossover methods for real numbers generate significant perturbations of the individuals such that it is questionable whether or not a mutation operator is actually necessary.

6.6 Comparative Performance against Linear Programming Methods

Also in consideration for effective means to solving the problem, a Linear programming method has been employed. This method has been used to determine the provable optimum performance on this problem.

It is common opinion that the GA should be much more effective at solving

this problem, particularly with more difficult instances of the problem. Also as the constraints become non-linear the GA is expected to outperform the LP.

At this first stage of the problem the results of the GA will be compared against the LP to determine the feasibility of the GA for this problem. As the problem becomes more complex the LP may be phased out completely should the GA live up to the high expectations that have been levied upon it.

7.0 Performance

After observing the running of many experiments it is obvious that this problem is well suited to a GA. Many factors have contributed to the superb performance of the GA on this problem.

One of the primary observations that has been made in this testing phase is that there is a very small margin and a real barrier between attaining a very good result and a near optimal result. Some of the charts shown below will not seem to show a significant difference between different decision points. When looking at the numbers directly it is obvious that there are many results which are below a certain point on the fitness spectrum and very few which are actually above that point.

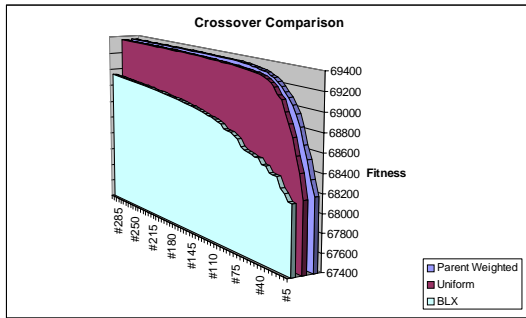
The difficulty in reading the graphs comes from the fact that the values are which is hard for the GA to attain are only a matter of 7 minutes different than those which are easier to attain. This significant difference is presented on an axes which is measured in 500 minute increments to show the trends of fitness over time, thus this significance appears insignificant.

7.1 Effective Crossover

Our expectations were met to a tee with respect to the crossover method. The parent weighted distribution contributed significantly to the overall success of this application of the GA.

Surprisingly, the uniform crossover method also performed quite well. This

Figure 7.1.1



could be due to the fact that these runs were executed with a large population size thus allowing the possibility that all necessary genetic material was available in the initial population. This was the case with all three random seeds which the runs were executed with. Even though the uniform crossover executed very well, the parent weighted method was a small yet significant margin ahead. This was seen to be the case in much of the testing. Several methods may have appeared to present similar results, yet there seemed to be some barrier to attaining the highest value possible. Only a few methods in combination were able to achieve the best overall performance.

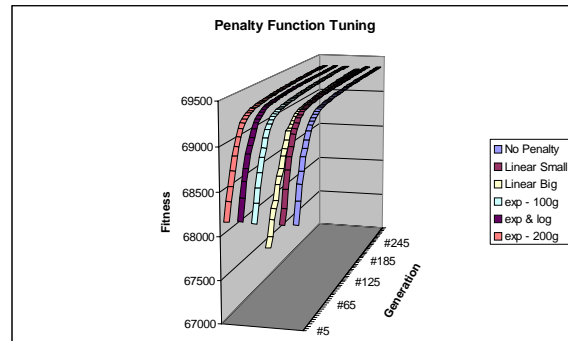
So, what we see here is that there are other options which will provide feasible results, yet the parent weighted distribution was the only one which was able to achieve that last step up in fitness.

7.2 Tuning of penalty function

Figure 7.2.1 displays the results of testing this GA with a set of different penalty function. This section will discuss this figure and relate other observations.

It is noted that without a penalty function at all the GA performed somewhat erratically. For a large portion of runs it would perform reasonably well generating feasible solutions, yet not reaching into the near-optimal space which we defined in section 7.0. Other times the GA without a penalty would perform very poorly seemingly unable to purge collisions from the population. These observations led to

Figure 7.2.1



the conclusion that there seemed to be a significant amount of natural pressure in this problem.

Two different versions of the linear formula (equation 4.2.1) were applied. This was done with a large constant value for C (130000) and with a small value for C (10). It turns out that the small C value tended to force out unfit individuals too quickly to achieve optimal results in the end. The small value performed much better yet it was still unable to attain the desired values due to existence of too much natural pressure.

To combat this, a new penalty function was applied of the form in equation 7.2.1. this equation sought to apply useful amounts of additional pressure at useful times in the evolution cycle. Thus it sought to apply little or no pressure in the beginning, and more pressure later on.

Equation 7.2.1

$$F = F_{pre} - (1 - ((e^{-\frac{g}{100L}})Cb))$$

In this equation F is the final fitness, F_{pre} is the fitness before any penalty is applied, L is a linearity factor, b is a constant weight to be multiplied with the number of collisions C.

This penalty function pressured collisions out of the population in an exponential fashion over time. The end result of this function was that a wider diversity of individuals was allowed to remain in the population for a longer period of time. After a point however it is determined that these individuals no longer

contribute positively to the fitness of the population and they are pushed out. The problem with the initial function which was applied was that it had a tendency to force out individuals with collisions much too early in the evolution process. Along with the bad blood which was forced out, genetic diversity was also forced out and thus lower levels of fitness were attainable.

The penalty function was also totally removed at one point and it was found that this GA maintained a significant amount of diversity and momentum yet it rarely was able to force out enough of the invalid individuals in order to get a valid solution.

Another function which was applied, seen in function 7.2.2, sought to force out individuals on a logarithmic scale with respect to the numbers of collisions. This function, combined with the exponential over time scale was unable to force out the last few collisions. There was a tendency for the last one or two collisions to remain in the population for a long time. It also was distinctive in that it forced out individuals with a large number of collisions very quickly. However, as stated before, the final solution usually was left with a single collision remaining.

Equation 7.2.2

$$F = F_{pre} - ((e^{-\frac{g}{100L}}) \log(C)b)$$

Figure 7.2.1 shows the relative performance of several different variations on each of the equations presented. The logarithmic equation (7.2.2) also had similar results. Several of the other runs performed quite well, at first glance, yet were often unable to generate a valid solution in the end. The only function that was able to routinely contribute to optimal and valid solutions was the exponential function (equation 7.2.1).

The surprise here was that the GA often performed adequately without a penalty at all. Often the bad individuals would succumb to natural pressures. However, this method tends to be less

reliable in that the rest of the time it tended not to generate a valid solution.

7.3 Useful Mutation

It was determined that mutation would not be tested as another variable to this problem. Particularly because the most favorable results were achieved with crossover methods which cause certain perturbations of the individuals and it seemed unnecessary to introduce more variability into the algorithm.

7.4 Comparative Performance against Linear Programming Methods

At the end of our testing it has become evident that the GA is quite capable of very nearly replicating the results of the LP. For a majority of the runs of the final tuned GA the final outcome easily approaches the provable optimum as determined by the LP. The GA however generates a wide range of perfectly feasible solutions well before the near-optimum is actually found. Depending on how strong a solution is necessary the GA could be allowed to run longer or could be cut short fairly early on in its execution.

The Linear programming method was allowed to run until it found what is expected to be a provable optimum of 69366.58. The tuned GA typically will return values that are 1-2 short of this value.

The Linear programming method had to run for approximately ten minutes in order to find the provable optimum. As shown in table 7.5.1 the GA ran for a comparable amount of time with a very large population. It is speculated that this time could be shortened significantly. One way might be to reduce the population size to some point between 200 and 500 as it is conceivable from the results presented in section 7.5 that the near-optimum values could be found with a population smaller than 500. Also it is observed that a run with a population of 500 very quickly moves into a space near the optimum and it is arguable that the entire population of 500 is not necessary for

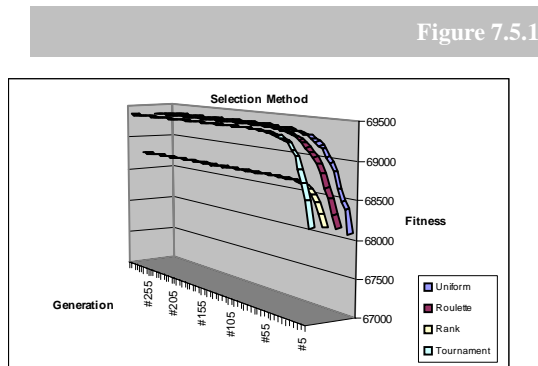
completing the fine tuning needed to arrive at the optimal value. It has been suggested that the population be reduced significantly part way through the evolution to speed the fine tuning process. While this was not tested it is left open for future work.

Another point that must be made while comparing the GA and the LP is that it was necessary in a few points to redefine the actual problem in order to maintain linear constraints. Otherwise the LP would not have been able to run. It is evident that the LP has a limited life span as the complexity of the problem increases.

7.5 Other Interesting Observations

There were other observations that were made while tuning this application. These involved the selection method being used and the population size that this GA was executed on.

Observation of the effects of the selection method contributed to the idea that significant natural pressure exists in this problem. One can quickly observe in figure 7.5.1 that the rank selection method performed significantly worse than even the uniform random selection method. Also notable is that the results from the tournament selection method were very similar to the results from the roulette wheel selection. As far as the end results of these two methods were concerned each beat the other equally many times as was beat by the other. In this particular series of runs it appears that the tournament selection method has a significantly steeper initial rise, however referring to the actual results



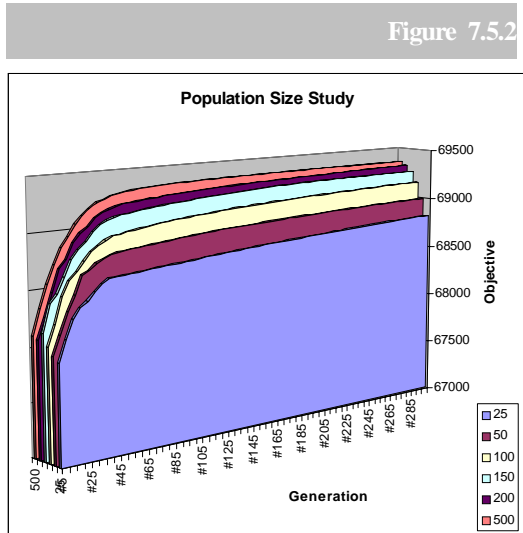
the two runs ended almost identically.

Another set of observations revolved around the setting of the population size. Several runs were executed each at different population sizes. Figure 7.5.2 suggests, as would be expected, that higher fitness is achievable with a larger population size. A couple of other facts must be used to judge how appropriate a particular population size is to this problem and the desired results. First, keep in mind that in order for a larger population to complete evolution of the set number of generations, it had to run significantly longer than a smaller population. See table 7.5.1 for details on running times.

Table 7.5.1

Population Size	Avg. Running Time
25	32 Seconds
50	1 Minute 4 Seconds
100	2 Minutes 15 Seconds
150	3 Minutes 10 Seconds
200	4 Minutes 10 Seconds
500	10 Minutes 40 Seconds

Whoever is interested in the results obtained by the GA must make the decision of what cost they are willing to pay for the results they get. It also has not been determined if the smaller population runs can achieve similar results to the larger populations. It is notable in figure 7.5.2 that the smaller population runs still seems to be steadily increasing as the end of the run while the larger population runs are not



increasing quite so obviously. It is questionable whether or not this would continue for much longer, and whether or not this could attain near-optimal results.

8.0 Future Work

There is a plethora of future directions that could be, and will be, taken from this work. Continued work on genetic operators is likely to continue as experimentation continues into different areas of the actual problem.

The problem also is to be expanded in future phases to include more constraints and more of those being of a non-linear nature. When this occurs it is expected that we will see a much more dramatic discrepancy between the linear programming method and the GA.

The immediate next phase of the problem entails the scheduling of the actual payload on the satellite. This scheduling will in a sense be the scheduling of what happens when the satellite is providing services to cities. The major obstacle in this case is the interaction between multiple satellites. Multiple satellites may be able to provide services to a particular city at the same time. The problem being, only a limited number of frequencies are available for providing these services. Therefore it is conceivable that the satellites could interfere with each other if not scheduled properly. Another factor that must be considered is the set of required service levels that may need to be maintained at each city. Just as multiple satellites may see a single city at one time, a single satellite may be able to see multiple cities at one time and must choose how to allocate it's resources in order to meet the service levels at each city.

Future work also will include the implementation of the core GA routines in Java to facilitate the visualization of the GA. Currently there exists a Java front end for visualizing the output of this GA, and of the LP, yet it is not as tightly integrated as it could be if it were implemented in Java.

9.0 Conclusions

This has been a fascinating application of the GA. Our client has been very pleased to learn about the potential benefits of using genetic algorithms on this project and others. The results obtained by the GA at this stage have been fully satisfactory and they point to excellent performance down the road when we begin to apply it to problems that are more non-linear in nature.

This approach of applying the GA to a problem which is being solved by a linear programming method and attaining comparable results has given the GA credibility on this project and viability for others.

This GA which has been applied was not as finely tuned as it could have been. Admittedly this project was more focused on the actual results attained by the GA then it was the actual inner workings of the GA. However since this project has gained some visibility for GA's in general we can assume that we will likely be able to concentrate more on the GA itself in the future as we seek to apply this technique to a variety of problems.

References

- 1 Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer 1992.
- 2 Goldberg, Genetic Algorithms in Search, Optimization & Machine Learning, Addison Wesley, 1989
- 3 F.Herera, M.Lozano and J.L.Verdegay, Tackling Real-Coded Genetic Algorithms: Pperators and Tools for Behavioral Analysis,
- 4 Matthew Wall, GALib, <http://lancet.mit.edu/galib-2.4/>