3-1994

A Genetic Algorithm Approach to Automating Satellite Range Scheduling

Donald A. Parish

AFIT/GOR/ENS/94M-10

A GENETIC ALGORITHM APPROACH TO AUTOMATING SATELLITE RANGE SCHEDULING

THESIS

Donald Arthur Parish
Captain, USAF

AFIT/GOR/ENS/94M-10

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

THESIS APPROVAL

STUDENT: Donald A. Parish, Capt, USAF CLASS: GOR-94M

THESIS TITLE: A GENETIC ALGORITHM APPROACH TO AUTOMATING SATELLITE RANGE SCHEDULING

DEFENSE DATE: 1 March 94

COMMITTEE:

Name/Title/Department	Signature
Advisor: James W. Chrissis, PhD, P.E. Associate Professor of Operations Research Department of Operational Sciences School of Engineering	
Reader: Gary B. Lamont, PhD Professor of Electrical Engineering Department of Electrical and Computer Engineering School of Engineering	
Reader: James T. Moore, Lt Col, USAF, PhD Assistant Professor of Operations Research Department of Operational Sciences School of Engineering	

AFIT/GOR/ENS/94M-10

A GENETIC ALGORITHM APPROACH TO AUTOMATING SATELLITE RANGE SCHEDULING

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Operations Research

Donald Arthur Parish, B.S.

Captain, USAF

March, 1994

Approved for public release; distribution unlimited

Acknowledgements

I would like to thank everyone who helped with this thesis. Of special note were my advisor Dr. James W. Chrissis for his solid recommendations, and my readers: Dr. Gary B. Lamont and Lt Col James T. Moore for their patience and understanding. I would also like to thank those whose work this research was based on: Capt Timothy D. Gooley, who was also my sponsor at AFIT, and Capt Stanley M. Schalck, who answered many of my questions about satellite range scheduling and data processing. Thanks are also due to my classmates and instructors who made the journey here an interesting one. Not least, I am grateful for the support and encouragement of my wife Lisa. I owe her much attention and a trip back to Nebraska.

Donald Arthur Parish

Table of Contents

Page

List of Figures

List of Tables

AFIT/GOR/ENS/94M-10

Abstract

Satellite range scheduling involves scheduling satellite supports in which a satellite and a specific remote tracking station communicate with each other within a specified time window. As the number of satellite supports continue to increase, more pressure is placed on the current manual system to generate schedules efficiently. Previous research efforts focused on heuristic and mixed-integer programming approaches which may not produce the best results. The objective of this research was to determine if a genetic algorithm approach to automating the generation of 24 hour schedules was competitive with other methods. The goal was to schedule as many supports as possible without conflict.

The genetic algorithm approach attempted to find the best priority ordering of support requests, and then used a schedule builder program to build schedules based on simple rules. A schedule was produced for seven days of representative satellite range data with slightly better results compared to earlier results using a mixed-integer programming formulation. Based on the reported results, the genetic algorithm approach presented in this research appears to be a competitive approach for generating 24-hour satellite range schedules.

A GENETIC ALGORITHM APPROACH TO AUTOMATING SATELLITE RANGE SCHEDULING

I. Introduction

Scheduling is the allocation of resources over time to perform a collection of tasks (2:2). Many important scheduling problems exist that are of interest to the Air Force. Examples include allocation of test range resources, airlift scheduling problems, and satellite communication support scheduling. Many standard scheduling problems may be formulated as mixed-integer programming problems where some decision variables take on integer values while other variables, such as time, have continuous values. While a mathematical programming formulation generally guarantees an optimal, or best possible, solution, the computation times required to find exact optimal solutions may be prohibitive for practical-sized problems. This is because many scheduling problems belong to the class of NP-complete problems (6:4).

In such NP-complete problems, the solution time may increase exponentially with the number of variables. For example, a problem with 10 times the number of variables as an original problem might take an order of 2^{10} times longer to solve. Because of these possible long solution times, heuristic techniques are often used to find good solutions in a reasonable amount of time. Heuristics are procedures which do not guarantee optimal solutions. They usually, but not always, provide feasible solutions, often require human ingenuity in their development, and are often quite problem-specific.

Heuristic methods for solving scheduling problems that are less dependent on the specific problem could be useful in solving problems of Air Force interest. This would be especially true when the constraints of the problem are difficult to put in mathematical form. One such method for finding good solutions to scheduling and other optimization problems is called a *genetic algorithm*.

Genetic algorithms (GAs) are artificial intelligence search methods based on the idea of natural selection and evolution. Initially developed by John Holland at the University of Michigan (10), applications include problems in optimization and machine learning. Although initially applied to function optimization problems, genetic algorithms have also been applied to scheduling and combinatorial optimization problems. The main strength of genetic algorithms is their ability to quickly explore a large space of possible solutions for good, if not optimal, solutions. They differ from many traditional algorithms as their search is based only on the overall evaluation of a set of parameters. As such, they do not need to rely on derivative information to proceed. Genetic algorithms are also able to find solutions where multiple optimal solutions exist (7:2-5).

Problem Description

One particular scheduling problem of interest to the Air Force is the ***Satellite Range Scheduling*** (SRS) problem. The Air Force Satellite Control Network (AFSCN) must schedule over 300 command and control communications per day between nine remote tracking stations (RTSs) and approximately 100 satellites (8:1–3). Each communication between a satellite and a RTS is called a ***support***. The satellite supports are necessary to maintain command and control, tracking, and system tests of the satellites (11:10). The scheduling of these supports must take place in a ***time window*** because each RTS is geographically separated from the others, and can only "see" each satellite for a limited time during its orbit. The time window is shorter for low-altitude satellites than for medium- or high-altitude satellites, as the low-altitude satellites pass out of the line-of-sight of the RTSs much more quickly than higher-altitude satellites. The longer time window of the higher-altitude satellites makes scheduling them less difficult than scheduling the low-altitude satellites.

Each Mission Control Complex (MCC) is responsible for the health, status, and orbital control of a subset of the total satellites. The composition of the subset depends on the mission type of the satellite. A MCC determines the length and required time windows for each support request for their satellites, as well as which RTSs can serve each request. These time windows may be more restricted than the physical visibility limits due to mission scheduling requirements. For example, a satellite may need a communication every hour after an operation is performed.

All requests from the individual MCCs are passed on to the range schedulers who are responsible for making an overall schedule. This schedule must ensure that each RTS antenna supports only one satellite at a time. The schedulers must also allow for a required turn-around time between supports to allow the RTS antennas to be reoriented. Down-time for RTS maintenance is also necessary. The resulting schedule is called the *initial 24-hour schedule*. The development of this schedule using genetic algorithms is the focus of this research.

The goal of the initial 24-hour scheduling process is to schedule as many communication supports as possible in a 24-hour period while satisfying the constraints of time windows, turnaround time, and scheduled maintenance down-time for the RTSs. If a conflict cannot be resolved by the range schedulers, they must de-conflict it with the MCCs and RTSs involved. If a support cannot be scheduled initially, it may be possible to facilitate its scheduling by changing the support requirements, by decreasing the turn-around time, or by altering other constraints. In the worst case, a support may not be scheduled. A good scheduling process minimizes the need for this deconfliction process. Although the current manual scheduling process schedules approximately 95-98% of the requested supports (8:5-2), the process is time-consuming, and could be streamlined by automating the development of the 24-hour schedule.

Previous Solution Efforts

Two recent thesis efforts have investigated automation of the 24-hour scheduling process. Gooley used both a mixed-integer programming (MIP) approach and heuristic scheduling methods to schedule satellite supports (8). As a follow-on research effort, Schalck improved Gooley's solution by reducing the number of integer variables necessary to define the problem (14). In both cases, the problem was too large to be solved in its entirety and had to be decomposed into smaller problems to find a solution in reasonable time.

Computational Complexity

Computational problems are often classified based on their complexity. According to Garey and Johnson (6:4), problems that are classified as NP-complete can take an inordinate amount of time to solve if the size of the problem is large enough. The general resource-constrained scheduling problem is in this class. According to Gooley, the SRS problem is a type of resource-constrained scheduling problem (8:2-9), and a fast (polynomial) solution cannot be guaranteed. Genetic algorithms, which have had some success in finding good solutions to other NP-complete scheduling problems, may be useful when applied to the SRS problem.

The mixed-integer programming approach can find optimal solutions for the satellite range scheduling problem when the number of support requests is small. However, when a larger problem is decomposed into smaller subproblems, a global optimal solution may no longer be guaranteed. A genetic algorithm approach may be expected to produce schedules that are at least as good while meeting the requirement of a short solution time since it would attempt to generate a solution to the entire problem. Also, a genetic algorithm approach can be more flexible in handling special case scheduling requests. These issues are addressed in this research.

Research Objective

The overall research objective is to determine whether a genetic algorithm-based solution methodology can effectively be applied to the satellite range scheduling problem. Successful solutions must:

1. Generate feasible 24-hour schedules which schedule the greatest number of support requests possible.
2. Find solutions quickly. A short solution time is useful in rescheduling as requirements change.

A secondary objective is to explore the scheduling of special-case requirements. These include supports which require simultaneous support on multiple RTSs and those that must be scheduled at fixed intervals. Rescheduling of satellite supports is also a consideration.

Overview

Chapter II is a summary of current literature relevant to genetic algorithm applications to scheduling problems and includes a review of previous work on the satellite range scheduling problem. Chapter III explains the methodology used in constructing a genetic algorithm solution to the satellite range scheduling process and details the implementation of the solution. Chapter IV presents the results and compares them to past efforts. Finally, conclusions and recommendations for future work are presented in Chapter V.

II. Literature Review

This chapter summarizes current literature on genetic algorithms (GAs) as related to scheduling problems. It begins with the development of genetic algorithms. Next, the extension of genetic algorithms to combinatorial optimization problems is reviewed. Discussion then turns to applications of genetic algorithms in scheduling problems, especially to problems with similarities to the satellite range scheduling problem. Finally, past research efforts in satellite range scheduling are discussed.

Genetic Algorithms

Introduction. Genetic algorithms were developed by John Holland as part of his artificial intelligence research on how artificial adaptive systems can evolve, or change, in response to their environment in order to solve problems (10). His ideas were based on the biological theory of evolution where variations in ***chromosomes***, or genetic codes, result in different traits of the individual. These traits of an individual in turn result in a level of performance, or ***fitness***, of an individual. Usually, the more fit individuals in a ***population*** survive from one generation to another and reproduce. Sexual reproduction by two individuals produces individuals with new chromosomes formed by a combination of the chromosomes of its parents. If a child inherits good parts of chromosomes from each of its parents, it has a higher level of performance than the parents. Through "survival of the fittest" and reproduction of high-fitness individuals in each generation, the population as a whole tends to evolve towards higher levels of fitness. The artificial version of this process is called a genetic algorithm. It does not seek to exactly simulate biological evolution, but the concepts of biological evolution are used to find good solutions to difficult problems.

Simple Genetic Algorithm. Most genetic algorithm work is based on the simple genetic algorithm as developed by Holland (10) and described by Michalewicz (13). Figure 1 illustrates the process and a brief review follows.

Coding. The first, and generally most difficult, step of any genetic algorithm is to choose a proper coding to map the problem solution space into a genetic string,

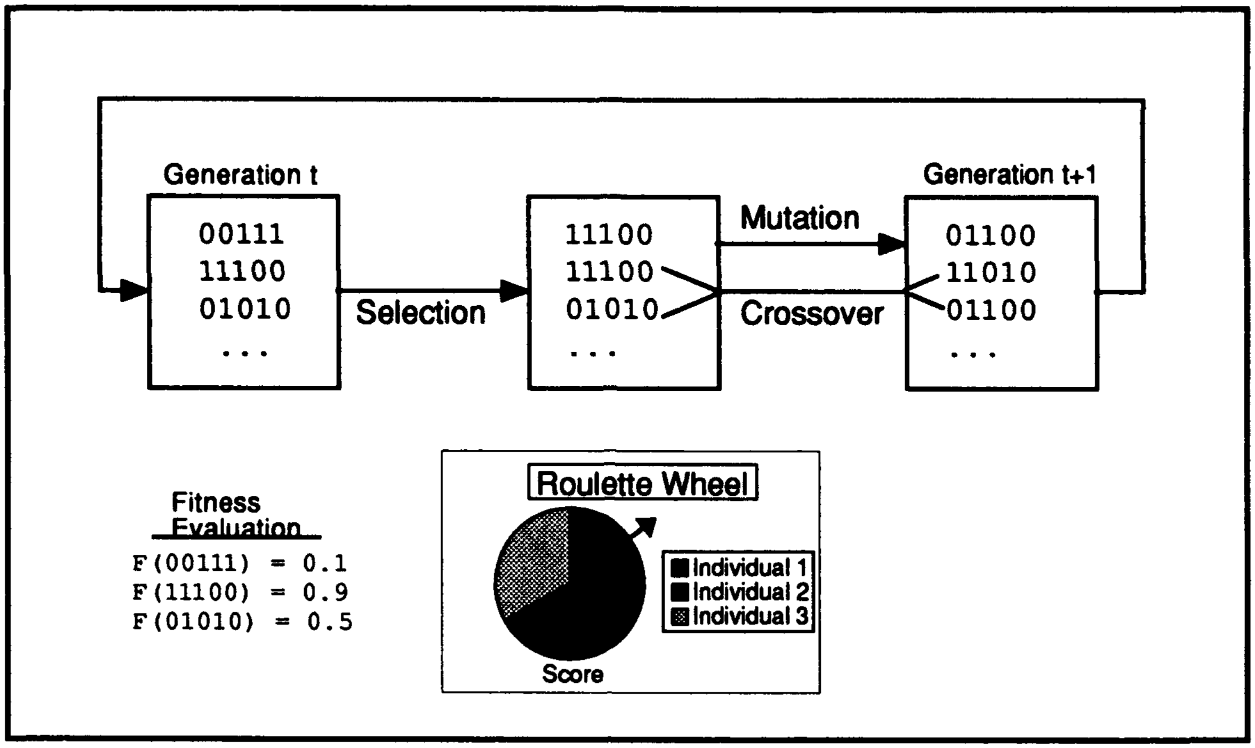


Figure 1. Genetic Algorithm Procedure

or chromosome, and to randomly create an initial population of individuals with varying strings. In the simple genetic algorithm, this coding is a binary string of zeros or ones. For function optimization, groups of binary digits are mapped so as to translate to a real number parameter representing the function value (13:19-20).

Evaluation and Selection. The strings in the population can be evaluated for their fitness relative to other strings in the population by entering their parameters into an evaluation or fitness function. The best strings reproduce by mating with each other to produce offspring for the next generation of the population. In the simple genetic algorithm, selection is governed by a "roulette wheel" selection operator. Each string has a probability of reproducing in proportion to the ratio of its fitness and the total fitness of the population. As in Figure 1, these ratios can be shown as pieces of a circular pie. New strings are selected for reproduction by randomly "spinning" the wheel. The strings whose proportion of the pie are greatest should, on average, be selected more often than the less-fit strings.

Other selection rules can be used (13:62). These include selection by ranking, where instead of a continuous scale based on fitness the strings are ordered, or ranked, by their fitness. The higher-ranked strings are given a greater chance of reproducing than average or low-ranked strings. In this way, the more fit individuals tend to reproduce.

Crossover Operator. During mating, two strings swap part of their "genetic" material. The children of these matings have parts of each of their parent's string. The swapping of genetic material, called *crossover*, allows for new child strings to be created, combining good aspects of their parents. The resulting strings with above-average fitness tend to survive and prosper, while those with below-average fitness tend to die out. An example of crossover is shown in Figure 2.

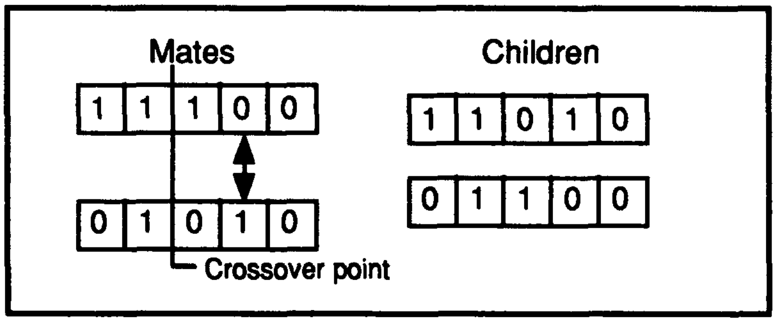


Figure 2. Example of Crossover

Mutation. Mutation randomly changes part of the string of a child to help maintain a diverse population. It is not as important as crossover in some applications since it merely acts as a type of random search (7:14).

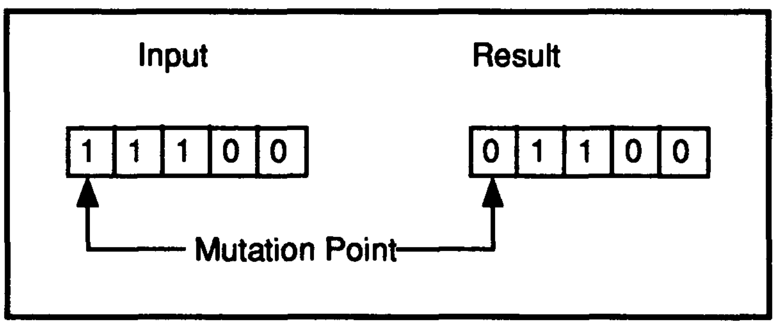


Figure 3. Example of Mutation

The general genetic algorithm thus consists of three steps which are repeated for each generation: evaluation/selection, crossover and mutation. These steps continue until termination conditions, such as some predetermined number of generations, are met. Often the genetic algorithm ends when the population converges: all strings evaluate to the same fitness (13:56). This string or strings may be the best answer, but a genetic algorithm may often converge to a suboptimum, in which case the population has prematurely converged. The major variables controlled by the experimenter to combat premature convergence are: population size, type and probability of crossover, type and probability of mutation, and selection operators (7:106-124).

Population Size. A genetic algorithm population contains a fixed number of strings. This number is called the *population size*. The population size affects the convergence rate of a genetic algorithm by controlling the variety of genes in the population. A smaller population may converge quickly, but usually to a sub-optimum. A larger population converges more slowly, and usually, but not always, finds a better final answer.

Although the basic procedure is simple, the framework of a simple genetic algorithm has been successful in solving a wide range of problems in function optimization and has been extended to other types of problems such as combinatorial optimization (13:165,193).

Why GAs Work. Although there is no complete formal theory to explain the operation of genetic algorithms, several hypotheses which partially explain their power have been advanced (13:51). The point they make is that in each generation, the genetic algorithm combines good partial solutions in the genes of parent chromosomes to find even better solutions in child chromosomes. The following discussion briefly describes these hypotheses. A more formal treatment of genetic algorithm theory is available in Goldberg's text (7).

Schemata and Schema Theorem. A *schema* is a pattern of values in a gene with the alphabet 1,0,*, where "*" is the "don't care" symbol matching any position. For a chromosome to include a particular schema it must match the schema values. For example, the chromosome (01101) contains 32 *schemata*. These include: (0**01), (*1****), (01101),

(*****), (0*1*1). The *defining length* of a schema is the distance between the outermost non-* symbols. The *order* of a schema is the number of non-* symbols contained in the schema.

Holland's schema theorem was the first rigorous explanation of how a simple genetic algorithm works. The schema can be thought of as representing the partial solutions in chromosomes and Holland concluded that genetic algorithms manipulate schemata when they execute (13:91). In the simple genetic algorithm, individuals reproduce and increase in proportion to their fitness. The schema theorem asserts that because of this, schema associated with individuals with above average fitness tend to increase exponentially, while those schema associated with below average performance tend to occur less often in succeeding generations (7:32-33).

Building Block Hypothesis. Much of the power of genetic algorithms comes from finding good *building blocks* (7:41). Building blocks are highly fit, low-order schema of short defining length. Because of their short length, these blocks tend to survive, even under the disruption caused by crossover. "In a way, by working with these particular schemata (the building blocks), we have reduced the complexity of our problem; instead of building high-performance strings by trying every conceivable combination, we construct better and better strings from the best partial solutions of past samplings" (7:41).

Together, the schema theorem and the building block hypothesis help to explain why genetic algorithms work. However, the hypothesis does not give a formula for designing genetic algorithms. Instead, most practical information on genetic algorithm design and performance have come from empirical studies.

Permutation Genetic Algorithms

The simple genetic algorithm with a binary coding is appropriate for many unconstrained optimization problems. However, in many *order-based* problems, the solution may be specified by a specific arrangement of items. Examples include scheduling problems and the Traveling Salesman Problem (TSP).

Traveling Salesman Problem. The Traveling Salesman Problem is easily stated: a salesman must visit customers in each of n cities without visiting a city twice (13:165-167). The objective is simply to travel the least total distance and return to the starting city. A sample network of cities is shown in Figure 4. The solution to the TSP can be represented as a list of integer numbers with each integer corresponding to a city, and the cities visited in the order of the list. For example, a starting solution for a TSP with six cities could be represented as: A = 1 2 3 4 5 6. This solution visits each city in ascending order from "1" to "6", returning to "1" to complete the tour. All possible solutions to the TSP can be

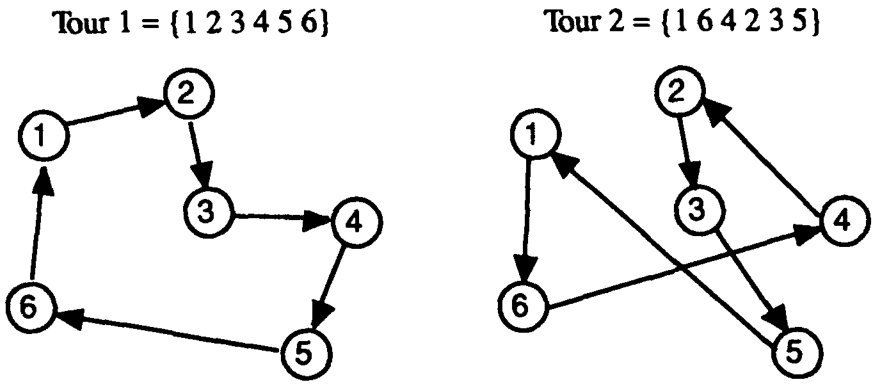


Figure 4. Sample TSP tours

represented as a permutation of the list of integers. A *permutation* is simply an arbitrary reordering of a set of items in a list. One permutation of the 6-city problem is A = 1 6 4 2 3 5. Although the ordering of the cities has changed, the number of cities remains the same and no city values are repeated in the list.

With n cities, there are $\frac{(n-1)!}{2}$ possible tours; a TSP with $n = 6$ has 60 ($\frac{5!}{2}$) possible solutions. For a TSP twice as large ($n = 12$), the total number of permutations is 239,500,800. The number of possible solutions is obviously incredibly large. Although smaller problems can be solved via deterministic graph search formulations, most large TSP problems are solved (non-optimally) using heuristic techniques which take advantage of distance information between the cities (12).

Blind Traveling Salesman Problem. Note that the TSP being solved by the genetic algorithm is harder to solve than the typical TSP because it does not use any distance information to solve the problem. "In the blind traveling salesman problem, the salesman has the same objective with the added restriction that he is unaware of the distance he travels until he actually traverses a complete tour" (7:170). However, other ordering problems may not have "distance" information that can be exploited by a heuristic. As pointed out by Whitley, "This is important, because it means the method may be used on sequencing problems where there are no actual distances to measure, but rather only some overall evaluation of the total sequence" (19:137).

Since genetic algorithms had been applied successfully to other optimization problems, it seemed natural to attempt a genetic algorithm solution to the TSP. However, when standard genetic algorithms are applied to the TSP, they have difficulty in finding good feasible solutions (9). The major problem with a standard crossover operator is that most solutions are infeasible (i.e., 1 2 2 4 5 6) and the genetic algorithm probably converges on sub-optimal solutions (13:167). Some early attempts used a hybrid technique to combine genetic algorithm crossover with a repair mechanism to make valid tours (9). This approach was somewhat successful, but other approaches have been attempted using order-based crossover operators (13:168-191).

Order-based Crossover Operators. Another approach to solving ordering problems uses a different representation than the standard genetic algorithm. Instead of binary digits, an *order-based* crossover operator uses a chromosome that directly represents a solution. This chromosome is simply a list of integers; an example is a solution to the six-city TSP problem described previously (1 6 4 2 3 5).

To exploit the information in the chromosomes, an order-based crossover operator, like simple crossover in the standard genetic algorithm, preserves part of the first parent while incorporating information from the second parent. The position of the genes is important. This contrasts with a standard genetic algorithm where their *value* is important. For the benefits of genetic algorithms to be realized for ordering problems, crossover operators must find other ways of combining information from two parents to build offspring with better fitness.

Although not strictly following the standard schema theorem for binary-based genetic algorithms, researchers have developed the concept of *ordering schemata* for order-based genetic algorithms (7:175–179). In an ordering problem, the absolute or relative *positions* of the items are important.

Ordering Schema. Ordering schema use a "don't care" symbol "!" to represent unfixed *positions* in a string. This is different from the "*" symbol presented earlier which represented unfixed *values* in a string. For example, in a *position*-based schema, the string (! ! 6 4 ! !) represents cities 6 and 4 in the third and fourth positions, with the "!" position filled arbitrarily from the remaining cities. Possible strings with this position schemata include: (5 1 6 4 2 3) , (2 5 6 4 1 3), (3 2 6 4 5 1).

Another ordering schema uses *relative* ordering. As described by Davis(5:79):

> It is important to understand that what is being passed back and forth here is not information of the form "node 3 is in the fifth position." Instead this operator combines information of the form "node 3 comes before node 2 and after node 7." The schemata in a *n* order-based representation can be written in just this way. Let us denote the nodes in a permutation by their indices. The chromosome (5 1 6 4 2 3) contains a number of schemata, including (5 4), (1 4 2 3), (5 6 3), and (5 1 6 4 2 3).

New crossover operators were developed that could work with ordering problems. Examples include Partially Matched Crossover (PMX) (7:154), position-based crossover (16:343), and edge-recombination (19). These methods attempt to retain the benefits of crossover while maintaining feasible solutions. Two of these operators are described below.

PMX. PMX was developed by Goldberg for use in solving TSPs. He describes the operator using a ten-city problem as an example (7:171). Each city is visited in ascending order; a sample permutation is: (1 2 3 4 5 6 7 8 9 10).

> Under PMX, two strings (permutations and their associated alleles) are aligned, and two crossing sites are picked uniformly at random along the strings. These two points define a *matching section* that is used to effect a cross through position-by-position exchange operations.
>
> To see this, consider two strings:

$$A = 9\ 8\ 4\ \boxed{5\ 6\ 7}\ 1\ 3\ 2\ 10$$

$$B = 8\ 7\ 1\ \boxed{2\ 3\ 10}\ 9\ 5\ 4\ 6$$

PMX proceeds by positionwise exchanges. First, mapping string B to string A, the 5 and the 2, the 3 and the 6, and the 10 and the 7 exchange places. Similarly mapping string A to string B, the 5 and the 2, the 6 and the 3, and the 7 and the 10 exchange places. Following PMX we are left with two offspring, A′ and B′:

$$A' = 9\ 8\ 4\ \boxed{2\ 3\ 10}\ 1\ 6\ 5\ 7$$

$$B' = 8\ 10\ 1\ \boxed{5\ 6\ 7}\ 9\ 2\ 4\ 3$$

where each string contains ordering information partially determined by each of its parents (7:171).

Position-based Crossover. Another crossover operator, ***position-based crossover***, was developed by Syswerda (16) and is called uniform order-based crossover by Davis (5). This operator attempts to preserve information about the relative ordering of elements in each of the parents. Several random positions are selected from each parent. These positions are inherited by one child. The other positions in this child are inherited in the order they appear in the other parent, skipping over all those included by the first parent. For example, using the strings used to illustrate PMX, first generate a bit string that is the same length as the parents:

```
A =  9 8 4 5 6  7 1 3 2 10
     0 1 1 0 0  1 0 0 1  0
B =  8 7 1 2 3 10 9 5 4  6
```

Next, fill in some of the positions on Child 1 by copying them from Parent A wherever a "1" appears in the binary template.

```
Child 1 =  - 8 4 - - 7 - - 2 -
           0 1 1 0 0 1 0 0 1 0
```

A list of the elements in Parent A associated with "0" is made, and permuted to appear in the same order as in Parent 2. These permuted elements fill in the gaps in Child 1 to complete crossover.

List of elements associated with "0": (9,5,6,1,3,10)

Permuted in order of Parent 2: (1,3,10,9,5,6)

Child 1 = 1 8 4 3 10 7 9 5 2 6

A similar process is used to create a second child, but with the roles of Parent A and B reversed.

Although PMX and position-based crossover are effective for ordering problems, they process different kinds of ordering schema. PMX tends to respect absolute city position, whereas position-based crossover tends to respect relative city position. Davis explains:

> The information that is encoded here, however, is not a fixed value associated with a position on the chromosome. Rather, it is relative orderings of elements on the chromosome. Parent 1 may have a number of elements ordered relatively well. Uniform order-based crossover allows Parent 2 to tell Parent 1 that others of its elements should be ordered differently. The net effect of uniform order-based crossover is to combine the relative orderings of nodes on the two parent chromosomes in the two children (5:79).

These crossover operators are mentioned because they were used during the implementation of this research. They are effective for problems where ordering is important. PMX is more important where absolute position matters, while position-based crossover is more effective where a relative ordering is more important.

Genetic Algorithms in Scheduling

Many recent papers involving genetic algorithms deal with scheduling problems. Two main approaches to solving scheduling problems with genetic algorithms have been developed: direct chromosome representation and indirect chromosome representation.

Direct Chromosome Representation. Bruns advocates a direct chromosome representation for scheduling problems. In a direct problem representation, the production schedule itself is used as a chromosome. No decoding procedure is therefore necessary. The extended chromosome representation requires the construction of domain-specific recombination operators (4:355) and would therefore be problem-specific. In this case:

> The sequence of the items within a chromosome is of no importance. To determine the quality of a chromosome any arbitrary evaluation function that has been used in traditional scheduling approaches is applicable without any prior transformation. (4:356)

According to Bruns, this approach should perform better than domain-independent operators, although he admits he cannot rely on any theory (4). It also involves creating custom crossover operators for each problem, which may be difficult. However, this approach has the advantage that the genetic algorithm is allowed to search the entire solution space, not just the ordering of the requests.

Indirect Representation. An indirect representation may also be referred to as a ***hybrid*** technique (7:202) where part of the problem is solved with the genetic algorithm and the other is solved using a deterministic routine. The first part is a sequencing problem, solved by the genetic algorithm, which orders each job request in a list. The second part is a schedule builder which takes each job, in the order of the list, and attempts to place each job request in a place in the schedule without overlapping another scheduled job. Obviously, those jobs earlier in the list are easier to schedule since more room in the schedule is available.

The schedule builder uses a simple rule to schedule each job; usually, a job is scheduled in the first position which meets the constraints of the problem. Although a schedule builder can use some information to find a good place in the schedule for each job, it is usually best to attempt the genetic algorithm search for good overall schedules by permuting the list of supports (16).

Much research on genetic algorithms in scheduling has been conducted recently in areas such as job shop scheduling and vehicle routing (3:452-459). For application to the satellite range scheduling problem, resource and sequence scheduling are more relevant. Two of these are reviewed by way of example in the next section.

Scheduling Examples

F-14 Test Range Scheduling. To solve a resource scheduling problem involving a test laboratory for F-14 fighter aircraft, Syswerda separated the genetic algorithm from the specific problem (16:332). The list of items to be scheduled is represented as a string of numbers. The genetic algorithm permutes the order of the items in this string to find the best order in which to schedule the items. Then, given the ordered list of items produced by the genetic algorithm, a schedule builder program builds a feasible schedule. The schedule builder is merely a program which attempts to schedule each item in the order presented by the string. The number of items successfully scheduled is returned to the genetic algorithm as a fitness score. This type of approach to scheduling assumes that given a correct ordering of tasks, the schedule builder program can build the best schedule. Syswerda noted:

> One thing that is clearly important, especially with regard to the greedy considerations of a single task, is the position of that task in the list. The closer the task is to the front of the list, the greater is its chance that it will be placed into the schedule ... if two tasks both require a scarce resource, the first task in the list may prevent the second from being scheduled, implying that the order of tasks is also important. (16:340)

Syswerda's implementation is interesting because he was able to satisfy many scheduling requirements such as priority of items and user preferences for scheduling days. Such flexibility is important for successful implementation of a scheduling solution.

Coors Scheduling. Much work has been done at Colorado State University on developing genetic algorithms for application to the Traveling Salesman Problem and some scheduling problems. Whitley developed a genetic algorithm called GENITOR (described in Chapter III, and used for this research), and Whitley and Starkweather developed an order-based crossover operator called *genetic edge recombination* for use in solving traveling salesman problems. Along with such theoretical developments, they also applied genetic algorithm solutions to a warehouse/shipping scheduler at Coors (15:74), and a production line scheduler at Hewlett-Packard (18:358–360).

From this research Whitley generalizes the application of genetic algorithms to scheduling problems in general:

> ... a broad class of scheduling problems can be viewed as sequencing problems. By optimizing the sequence of processes or events that are fed into a

simple schedule builder, optimization across the entire problem domain can be achieved. Schedules have not always been viewed this way because there has not existed a general purpose mechanism for optimizing sequences that only requires feedback about the performances of a sample sequence... Thus, a "genetic" approach to scheduling has the potential to produce some very general scheduling techniques, and could be the foundation of a general purpose approach to sequence scheduling. (18:358)

For this research, an approach similar to that used by Syswerda in the F-14 test range scheduling problem seems promising, as it does not require a custom chromosome. Hence, existing genetic algorithm packages, such as GENITOR, may be used for implementation. This approach only requires selecting an ordering crossover operator and constructing a schedule building program to simulate the scheduling operation. As described in the next chapter, such a strategy can be easily formulated.

Other Solution Efforts for Satellite Range Scheduling

Solutions to the satellite range scheduling problem have been studied recently. These include an effort by IBM, and two thesis efforts at the Air Force Institute of Technology (AFIT). These efforts used mixed-integer programming and heuristic approaches to find solutions.

Arbabi's Approach. The first study of automating satellite range scheduling took place during the 1981-84 time period when IBM conducted a study to determine the feasibility of automating satellite range scheduling (1:271–277). Arbabi concluded that a mixed-integer programming approach was not feasible for problems with more than 50 requests. Instead, he developed an approach called Continuous Time Scheduling (CTS). The procedure was not described in detail, as it is apparently proprietary, but it used a heuristic approach. This procedure reportedly scheduled 92% of the requests for one day (1:277).

Gooley's Approach. Gooley used both a mixed-integer programming approach and heuristic scheduling methods (8). The satellite range scheduling problem was successfully formulated as a mixed-integer program (MIP), but the number of integer variables prohibited direct solution. To reduce the number of integer variables, Gooley divided the problem into two mixed-integer programs which scheduled low-altitude satellite supports (the MIP could handle up to 85 low-altitude supports at once). He then used heuristic insertion and interchange techniques to schedule the medium- and high-altitude satellite supports. Using this method, a test set of problems was solved in under 20 minutes with approximately 92% of all requested supports scheduled (8:5-2).

Schalck's Approach. In a follow-on thesis effort, Schalck improved on Gooley's solution by reducing the number of integer variables needed in the MIP formulation. By doing so, one MIP solution scheduled all low-altitude supports for one 24-hour period. Solution times for scheduling the high-altitude satellites in one 24-hour block was too long, and was reduced to about 30 minutes by scheduling high-altitude supports in two 12-hour blocks. The resulting solution scheduled approximately 98% of requested supports (this number is not directly comparable to Gooley's results because of differences in how the support requests were generated).

The mixed-integer programming approach taken by Gooley and Schalck is the best approach for the satellite range scheduling problem when the problem is small enough to be solved by the mixed-integer program, since such an approach should find an optimal solution. However, by decomposing the problem into separate problems, an overall optimal solution may no longer be guaranteed. For example, splitting the requests into two blocks means that support requests near the division point may not be scheduled. A genetic algorithm approach may produce better schedules while meeting the constraints of a short solution time because it would attempt to find a solution to the entire problem at once. Although the genetic algorithm approach is not guaranteed to find an *optimal* solution, it can attempt to find good solutions. Also, a genetic algorithm approach may be more flexible in handling additional constraints for special scheduling requests.

Summary

This chapter summarized the development of genetic algorithms including the standard genetic algorithm and variants. The extension to order-based genetic algorithms allows good solutions to some combinatorial optimization problems, such as the TSP. This in turn allows applications of genetic algorithms to scheduling problems by dividing the algorithm solution between a deterministic schedule builder and a genetic algorithm. Finally, past research efforts in satellite range scheduling were reviewed, with the conclusion that better solutions may be found by scheduling the entire day's schedule in one time block instead of decomposing the problem into smaller time blocks. This may be done with a genetic algorithm based approach.

III. Solution Methodology for Satellite Range Scheduling

This chapter presents a review of the satellite range scheduling (SRS) problem, followed by the development of a genetic algorithm-based scheduling strategy. By formulating the scheduling problem as a sequencing problem rather than as a mathematical program, a straightforward solution by an order-based genetic algorithm is possible. The chapter ends with a review of the implementation of the approach using the genetic algorithm package GENITOR.

Formulation

Definition of the Problem. In the SRS problem, satellite communication ***supports*** compete for RTS (remote tracking station) time in a 24 hour schedule. Each support must be scheduled in a restricted time window at certain RTSs due to visibility and scheduling requirements. For the low-altitude satellites, the requested support length fills the entire window. For medium to high altitude satellites, the time window is more flexible, as a tolerance for the beginning of the support is allowed for scheduling. Although each low-altitude support requires a fixed time window at one RTS and fills a time-window, the scheduling task is eased as most RTSs have two ***sides*** (antennas) capable of supporting communications. The RTS sides can support a satellite support simultaneously; if a satellite is visible to an RTS, it can be supported by any one of the available sides at the RTS. In addition, most medium-to-high altitude satellites are visible to more than one RTS. This fact, combined with the more flexible time windows of these satellites, makes scheduling them easier than scheduling low-altitude satellites.

Small Problem. An example set of time windows for a small set of five support requests is shown in Table 1. "Spt" is an arbitrary support number, "Begin" is the starting time for the window, "End" is the ending time for a window, "Length" is the actual service time needed, and "TAT" is the setup time required before a service time can begin. These supports are all serviced by RTS "POGO-A."

This sample time window data can be used to illustrate the satellite range scheduling (SRS) problem. This small problem is simplified since a real day's schedule would include

S[illegible]	Begin	End	Length	TAT
1	1	13	3	1
2	15	22	3	2
3	7	17	3	1
4	1	10	3	1
5	2	8	3	2

Table 1. Small Schedule Time Windows

over 300 supports, nearly all of which would have alternate windows for scheduling. These alternate windows could come from different antennas or sides at the same RTS, or from different RTSs. In addition to the table, this information is shown graphically in Figure 5.

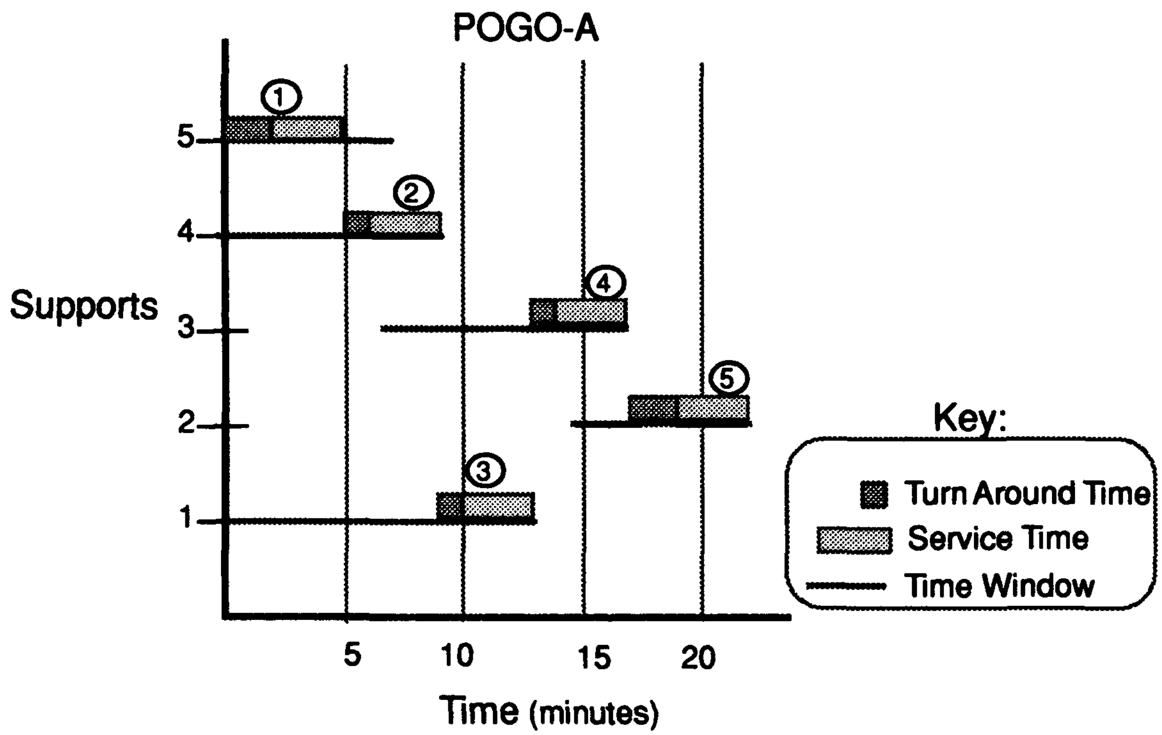


Figure 5. Small Schedule

The horizontal axis shows time in minutes, the vertical axis shows the supports by support number, and the entire chart is for RTS POGO-A. A problem with more than one RTS could be represented by more than one chart. In Figure 5, the *time window* is represented by a thin line, while the required TAT and service time is shown by the boxes (darker pattern for TAT). Such a representation clearly shows a schedule. The schedule

for the example successfully schedules all supports with no overlap of the support times and no violation of time windows. The TAT for support 5 is outside the service support window, but this is legal as no communications take place at that time, only set up for a support.

Mixed-Integer Programming Approach. Previously, the SRS problem has been formulated as a mixed-integer programming (MIP) problem (Gooley and Schalck). For a problem with a small number of variables, this is the best approach because an optimal solution can be found. But, as the number of supports to be scheduled increases, so does the number of integer variables in the MIP formulation. Because of this increase, an optimal solution may no longer be available in a timely manner. Gooley and Schalck address this problem by decomposing the problem into parts. Gooley combined the use of a MIP formulation with heuristics. Schalck accomplished variable reduction and scheduled the satellites using an MIP for various combinations of satellites and blocks of time. These methods produce feasible schedules which are not necessarily optimal.

A MIP solution includes the starting time for each support and the RTS used for the support. The schedule can then be generated by adding the service times to the starting times. With this type of formulation, the mathematical program must find the actual starting times without violating constraints. A MIP solution of the small problem described in the previous section might begin with support 1 scheduled to start at time 10, support 2 scheduled to start at time 19, and so on, as shown in Figure 5.

Scheduling as a Sequencing Problem. Instead of formulating the satellite range scheduling problem as a mixed-integer program, this research approaches the problem as a *sequencing* problem where a sequence is defined as an ordered list of items. A solution to a sequencing problem in scheduling determines the "best" order in which to schedule items using simple rules for placing each item in the schedule. For example, in the satellite range scheduling problem, the solution is represented as a sequence of supports. In the small problem, there are five supports. An example ordering is: (5, 4, 1, 3, 2). To build a schedule from this representation, each support is scheduled according to the order it appears in the list.

In the example schedule, the first support, 5, would be placed in the first available time segment. The entire schedule is open and support 5 is scheduled at the beginning of its time window from time 2 to time 5. Note that the turn-around time of two minutes is scheduled from time 0 to time 2 which is outside the time window. This is valid since no communications take place during the turn-around time. The schedule is then updated to reflect that time 0 to 5 is no longer available. The second support in the list is 4. Its time window begins at 1, but it cannot be placed there because support 5 has already used minutes 0 to 5. Support 4 is then scheduled from time 6 to time 9 (1 minute TAT; 3 minutes for service). This support remains within its time window which ends at time 9. This procedure repeats for supports (1, 3, 2), and completes a schedule as shown in Table 2.

Spt	Begin	End	Length	TAT
5	2	5	3	2
4	6	9	3	1
1	10	13	3	1
3	14	17	3	1
2	19	22	3	2

Table 2. Small Problem Schedule

The actual start and end times for each support were determined by examining the completed schedule. Thus, all the information needed to define a solution is contained in the sequence of supports, and can be "decoded" by following simple deterministic rules. Obviously, such rules are somewhat arbitrary. The "first available" rule, as shown here, is simple. A rule which attempts to find the "best" place for each particular support would also work, but at the expense of simplicity and possibly time.

An arbitrary ordering of supports is not likely to produce a perfect, or even good, solution which schedules all support requests. However, if a solution exists, an ordering can be found which results in the optimal solution. In many cases, more than one ordering results in the same schedule depending on the interdependence of each support request. To find good schedules, an efficient way of generating better alternative orderings is needed. In this research, a genetic algorithm with order-based crossover operators is used to spawn good orderings of the support requests. Genetic algorithms can quickly search the solution space, defined as the permutations of the list of items. These permutations can then be used to build a schedule as described above, and the number of supports scheduled successfully can be used as a fitness measure for the genetic algorithm search.

Hybrid Approach. As noted in the previous section, a schedule can be built from a sequence of supports using a schedule builder program. This schedule builder is required for building *feasible schedules* given a sequence of support by the genetic algorithm. The overall process is shown in Figure 6. Although some decision information can be incorporated into the schedule builder to improve local search, the genetic algorithm should do most of the exploration of the search space. This approach is based on the assumption that if the supports are entered in the schedule builder in a certain order, the greatest number of supports can be scheduled. It is reasonable to expect that many orderings of the supports may exist which produce the same schedule.

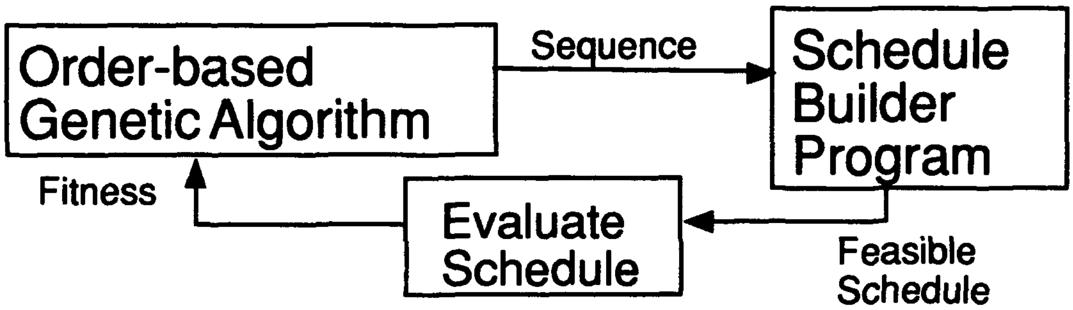


Figure 6. Schedule Builder Flowchart

The nature of the genetic algorithm and schedule builder approach allows other constraints to be imposed on the supports; for example, a request that a support be handled by a certain set of RTSs on a particular day, or that supports must be scheduled exactly one hour apart.

Algorithm Design and Implementation

Overview. Implementation of a hybrid approach requires data preprocessing of satellite requests to generate support request time windows, followed by application of the

GENITOR genetic algorithm. The schedule-building evaluation function is used within GENITOR to produce a given schedule, given a sequence of supports from a GA string.

Assumptions. The research assumes the constraints of the satellite range scheduling problem are those used in the previous efforts by Gooley and Schalck (14:1-7):

1. Requested support times are known in advance.
2. Time windows for satellite visibility are not flexible. The goal of the SRS problem is to schedule as many supports without having to change the time windows. After the initial schedule is formed, these constraints may be relaxed to schedule unscheduled supports. However, it is desired to minimize the number of such changes.
3. Downtimes for RTS maintenance are not included. If known, flexible downtimes could be added as additional supports. Fixed downtimes could be be scheduled before regular supports are scheduled.

Use of these assumptions allows comparison of the genetic algorithm results to previous efforts.

Data Processing. Time windows for each support are processed from the raw satellite visibility and request data in the ASTROS database for a day. The support requests and time windows are represented as lines in a database with the following information:

Support Number: arbitrary support number

RTS: Remote tracking station and antenna side (example: POGO-A for A-side, POGO-B for B-side)

Beginning of Time Window: in minutes (example: 0200 is 120 minutes; 2400 is 1440 minutes)

End of Time Window: in minutes

Support Length (minutes): actual service time needed

Turnaround Time (TAT): set up time at RTS needed before service time (20 minutes for low-altitude satellites; 15 minutes for medium-high altitude satellites)

Satellite Identification: IRON (first four digits; identifies satellite)and revolution number (last three digits; identifies orbital pass)

An example set of time windows is shown in Table 3. This data is the same as for Table 1, but with fields included for the RTS name and and satellite identification number.

Spt	RTS	Begin	End	Length	TAT	Ident
1	POGO-A	1	13	3	1	2532097
2	POGO-A	15	22	3	2	4774042
3	POGO-A	7	17	3	1	9845009
4	POGO-A	1	10	3	1	3187074
5	POGO-A	2	8	3	2	9757024

Table 3. Small Schedule Time Windows

This time window data is read into data structures at the beginning of a run.

Data Structures. Time window data is read into a structure of arrays. The information can then be used each time a schedule is built from a sequence of supports. These array structures keep track of time window alternatives for each support, and allows the schedule to be filled in as supports are scheduled.

Schedule array. An array named *filled(RTS, time)* shows the status of each minute (time) for every RTS and antenna combination. A '0' represents an empty minute; a '1' indicates that minute is filled. This array is updated whenever a support is scheduled by setting the minutes used by the support to '1.' Any set up time is also blocked out of the schedule.

Support Information. Three arrays specify the requirements of each support: *NumWin(support)* is the number of time windows where a support can be potentially supported. Then the support time length requirements and turn around time are given by *Length* and *TAT*, respectively. This information is used when running the schedule builder program.

For each alternative time window for a support, three arrays give specific information. BVIS and EVIS give the beginning time and ending time, respectively, of each time window. RTS specifies the particular RTS used for the support.

Schedule Builder Program. The schedule builder is implemented as a program in the C language since many genetic algorithms are written in C. It is used as an evaluation function for the genetic algorithm by building a schedule from a sequence of supports generated by the GA, as shown in Figure 6. To construct the schedule builder, an insertion program was written, modeled loosely on those used by Gooley (8), which attempts to schedule each support into an available time window. Many alternative schedules are possible, as the insertion rules for the schedule builder are arbitrary. For example, the simplest implementation attempts to schedule supports to the first available numbered RTS. A schedule could also be built by assigning a support to an available RTS with the most open space left in its schedule. The "first available" approach was chosen for simplicity, and to schedule supports next to each other to utilize the available time at the RTSs. This approach simply schedules satellite communication supports to the first available position in time, and across each RTS, starting with the first RTS.

Each support is scheduled in the order it appears in the ordered list of support requests. The schedule builder tries the first RTS where there is a window for the support. If this fails, it tries the next until it runs out of windows. If the support *is* scheduled, the schedule score is incremented and the space used is blocked out of the schedule. Pseudocode for the program is as follows:

```
Empty schedule
Attempt to schedule each support
    Try each window until succeed or exhaust windows
        Try each set of empty spaces until succeed or reach end
        Update schedule if support scheduled
    Update score if support scheduled
Output final score
```

This can also be represented by a flow diagram, as in Figure 7.

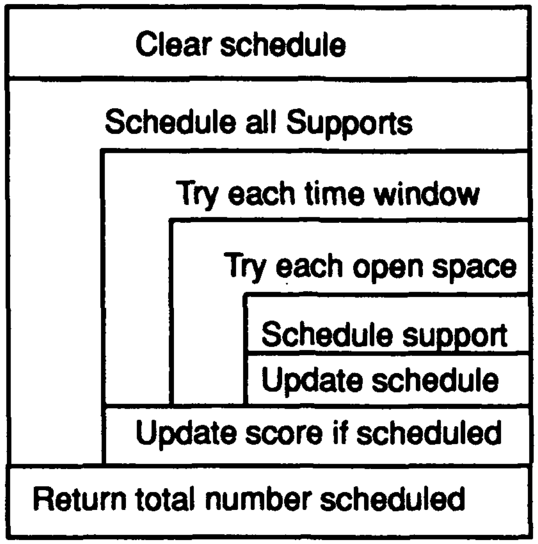


Figure 7. Schedule Builder program flow

This procedure finds the schedule corresponding to a sequence of supports. It is also used to build a schedule given a sequence of supports or to evaluate a given sequence as part of a genetic algorithm implementation.

Genetic Algorithm Implementation. To find "good" support sequences, the schedule builder is integrated into a genetic algorithm as an evaluation function. The genetic algorithm, GENITOR, and associated parameters are described,

GENITOR. Whitley's GENITOR (GENetic ImplemenTOR) code was chosen for use in this research because it has been used with success for other scheduling problems (17). GENITOR includes crossover operators for order-based genetic algorithms (edge recombination, order, and PMX) which have been shown to be useful in solving many scheduling problems (15). GENITOR differs from the standard genetic algorithm as described in Appendix A.

The evaluation function assigns a raw fitness measure to each population member. In GENITOR, the fitness is assigned by an evaluation function written in the C programming language (17). The function input parameter is the chromosome of a population member which is a list of supports. After attempting to schedule each support in the schedule builder, the function returns a fitness value. In the satellite range scheduling problem, the chosen fitness measure is the number of supports successfully scheduled.

GENITOR Parameter Settings. In GENITOR, the major settings are the crossover operator, population size, selective pressure, and mutation rate (for binary-encoded chromosomes):

Crossover. The crossover operator is important in the quality of the answer. GENITOR includes crossover operators for order-based genetic algorithms as discussed in Chapter II. These include edge recombination, order, and PMX.

Population Size. As explained in Chapter II, the population size affects the convergence rate of a genetic algorithm by controlling the variety of genes in the population. A smaller population may converge quickly, but usually to a sub-optimum. A larger population converges more slowly, and usually, but not always, finds a better final answer. For example, a population size of 30 finds a good answer within 1000 reproductions, but converges to a worse solution.

Selective Pressure. Selective pressure is a selection parameter specific to GENITOR. This parameter controls the rate at which a population converges by giving more reproductive opportunities to higher ranking individuals in the population. In GENITOR, selective pressure is usually set between 1.0 and 2.0. For example, a selective pressure of 1.5 gives the top-ranked individual 1.5 times the chance to reproduce than the median-ranked individual. A lower setting slows down convergence, hopefully allowing more time for the best solution to emerge. A higher setting drives the GA towards the final answer more quickly, but often at the expense of the best solution.

Mutation. It is important to keep a diverse genetic pool as the population converges. Mutation randomly changes part of a chromosome and is usually of less importance than crossover. Mutation was not used in this research because an order-based mutation operator was not present in GENITOR. The addition of such an operator may improve results and is discussed in Chapter V.

Day One Results. Various population sizes and selective pressure settings were tested in GENITOR using the first day of data to determine good parameter settings. In preliminary tests, all order-based crossover operators in GENITOR were tested. Position-based crossover found the best answer more quickly than the others. Position-based crossover was expected to produce good schedules since it explores the relative position of supports. Since supports are competing for resources, it makes sense that the relative order of supports in a scheduling list is more important than an absolute position in the list. Based on these results, the position operator was chosen as the operator for the test set, although the other order operators would be expected to perform nearly as well. The edge-recombination operator is tailored more towards the TSP and did not do as well as the other operators in this scheduling problem because it stresses adjacency information instead of relative ordering. This agreed with past results using edge-recombination for scheduling problems (15:74).

For Day One data, a graph of performance is shown in Figure 8. As GENITOR executes, at a given interval of reproductions it displays the best individual, the worst individual and the population average. The population has converged when these three numbers are equal, and little improvement can be expected. Convergence indicates that the individuals all have the same fitness score; they are either identical or similar enough to evaluate to the same fitness.

At any time during the run, the GA can be stopped and the best individual found thus far can be chosen as the best schedule found. Usually, the best schedule is found before convergence, but there is no way to know this in advance. In the Day One data, the best schedule was usually found at about 4000 reproductions, while the population did not converge until about 6100 reproductions. A genetic algorithm run could be stopped prior to convergence if time is critical.

Vary Parameters. The runs shown in Figures 9 and 10 display the interplay of exploitation versus exploration in a genetic algorithm solution (7:37). A high selective pressure or a small population size leads to quick convergence of the population. This exploitation of the best individuals found so far often leads to a quick, but less satisfactory

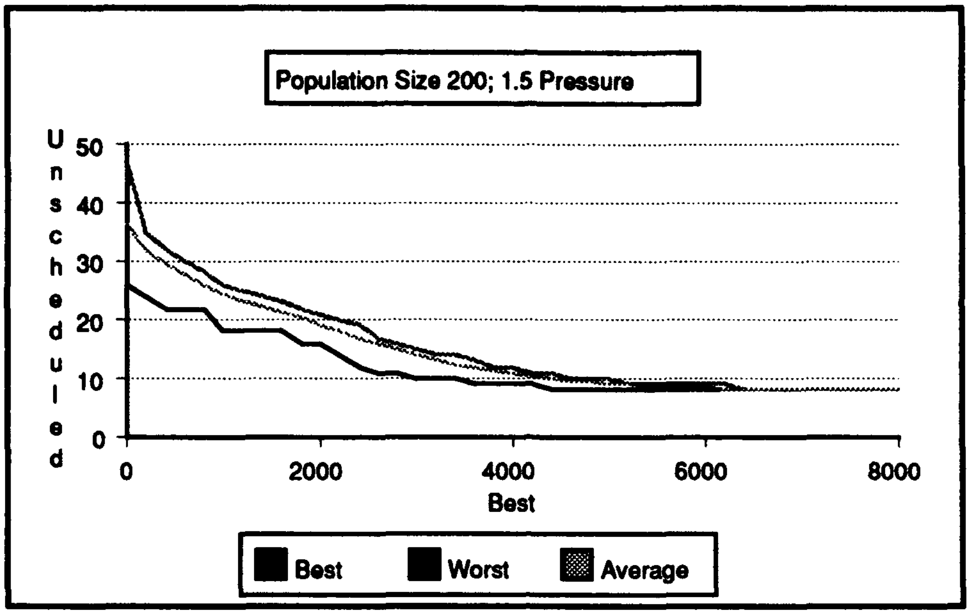


Figure 8. Day 1 Data Convergence

solution. On the other hand, a larger population with lower selective pressure encourages population diversity longer, but at the cost of slower convergence.

Repeated Runs. Since a GA has random components, a set of 10 runs was used for the Day One data in order to initially evaluate the variance of the results. Of 322 supports requested, the number of unscheduled supports was 8 in five of the runs and 9 in the other five runs; the average is 8.5, with a variance of 0.28. For this problem, the difference between runs is minor, and in practice one run should be sufficient for scheduling.

Random Schedule Generation. To give a baseline of the performance of this work, one might ask how well a random permutation of supports would perform. To answer this question, an additional run which generated random permutations of the support list and then used the schedule builder program to create schedules was performed. It found reasonable results within 8,000 reproductions, with a best solution of 23 supports unscheduled out of 322. This is not as effective as the best genetic algorithm schedule of only 8 unscheduled. These results are shown in Figure 11.

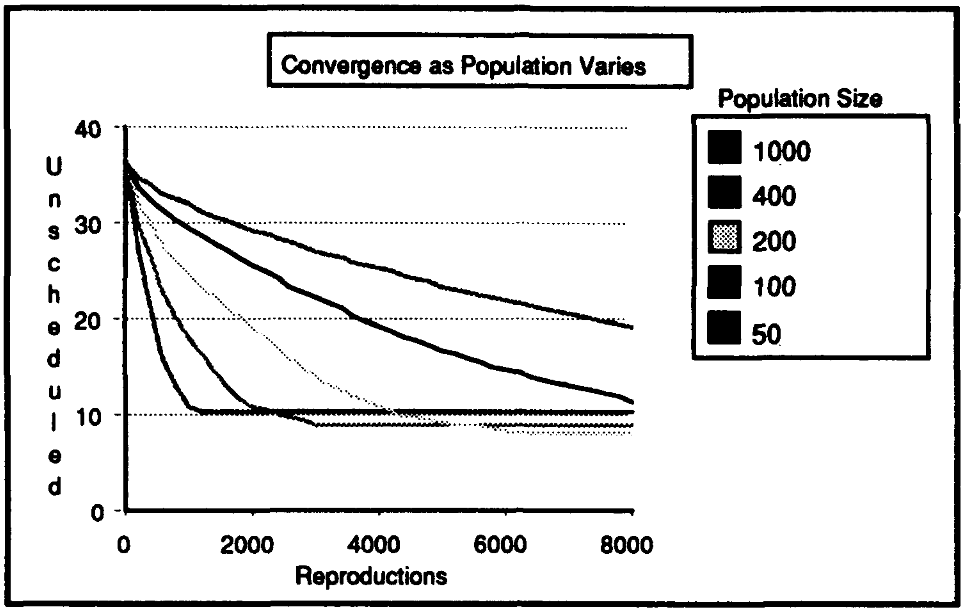


Figure 9. Vary Population Size for Day 1 Data

Overall Implementation. The overall implementation of the genetic algorithm-based approach begins with the processing of raw satellite request and visibility data into a format readable by the GENITOR main program. This processing is handled by Pascal programs written by Gooley and Schalck. After processing, the time window and request data for each support is stored in an array structure which the schedule builder uses to build feasible schedules.

Summary

This chapter describes how the SRS problem may be solved by scheduling items in an order determined by a genetic algorithm. By using this evaluation of each schedule to determine fitness in an order-based genetic algorithm, better schedules can be developed. The genetic algorithm package chosen for this research is GENITOR, because of its success in similar problems. Experiments using the first day of data provide good parameter settings for the GENITOR genetic algorithm in producing 24-hour schedules. The final settings are: position-based crossover, population size of 200, and selective pressure of 1.5. These settings gave a good compromise between the greatest number of supports scheduled and execution time. These settings are used to produce the schedules in Chapter IV.

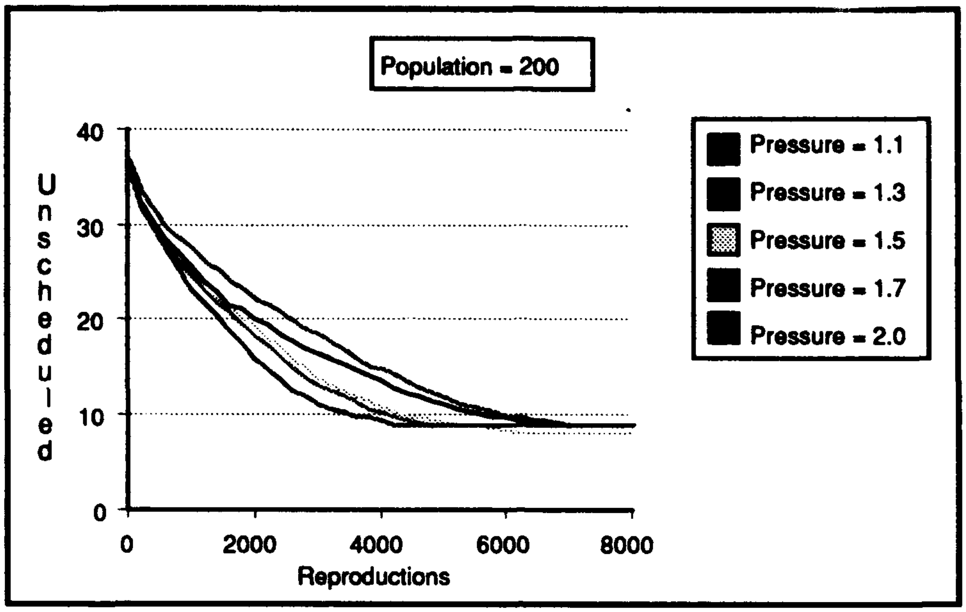


Figure 10. Vary Selective Pressure for Day 1 Data

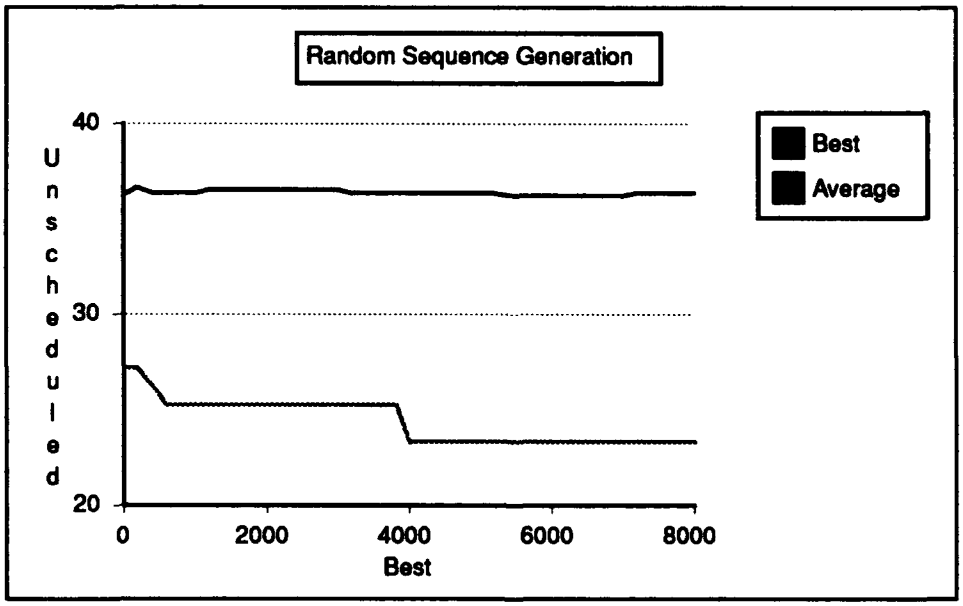


Figure 11. Random Sequence Generation for Day 1 Data

IV. Results

In order to evaluate the performance of a genetic algorithm approach to satellite range scheduling, the schedule builder model described in Chapter III was integrated into the GENITOR (17) genetic algorithm. This genetic algorithm was then tested on real satellite range data. In nearly all cases, the GA-based solution was able to match or exceed previous results that used a mixed integer programming (MIP) approach.

The primary purpose of this research was to evaluate the performance of the GA-based solution in scheduling satellite supports. A secondary purpose was to explore the flexibility of the GA-based approach in handling additional constraints on the scheduling of supports. Such flexibility would be important in implementing a genetic algorithm-based approach in the real world. The additional constraint tested here is the priority of different supports.

The criteria used to judge scheduling results is the number of supports scheduled and the time required for the solution. Supports not scheduled in the initial 24-hour schedule by the range schedulers must be deconflicted through coordination with the MCCs. In the worst case, those supports that cannot be rescheduled by relaxing constraints may have to be canceled. Short solution times are important if different parameters or priorities are to be used by the schedulers. Short solution times are also important for rescheduling on short notice.

Test Set

Scheduling data used in previous research efforts was available and used to test the hybrid GA approach, and to allow a comparison between these results and results of previous efforts. This data is taken from the ASTRO (Automated Scheduling Tools for Range Operations) (1:271) database which is currently used to assist manual scheduling. The primary data is from the seven day period from 12 Jul 92 to 18 Jul 92. This is the same data used by both Gooley and Schalck. Each day includes approximately 300 support requests and their associated request windows. Of these, approximately half are low-altitude satellite support requests.

The database contains satellite support requests and the visibility windows of the requests to remote tracking stations for each day. Pascal programs developed by Gooley and modified by Schalck were used to to put the requests in a format suitable for further work. Schalck's processing of the data was used here to allow comparisons to his results. After processing, the final data tables list the visibilities and requests for each day as shown in Table 3 of Chapter III. Processing details are available in Appendix D.

Although each day is scheduled separately, supports are allowed to overlap into the next day if their time window extends into the next day. This is included to match the approach of Schalck. The genetic algorithm implementation schedules across days by keeping track of the overlap into the next day and scheduling these overlaps in the next day's schedule before scheduling regular supports.

The schedule builder code developed in Chapter III serves as the evaluation function in GENITOR. The schedule builder produces a valid schedule from an ordered list of supports for each string in the population. The number of supports successfully scheduled is the fitness of each string. This fitness value is then returned to the genetic algorithm. The fitness is used by GENITOR to rank a particular schedule in relation to others in the population. The more fit strings have a higher probability of being selected to reproduce and exchange their genetic information with other strings through crossover.

Experimental Procedure

Tests were conducted in two major stages. The first day's data was tested to find good parameters for the GA (as shown in Chapter III). These parameters included population size and selective pressure. The primary goal of these parameters is to facilitate achievement of the best solution within the shortest time, where the solutions are scored by the number of supports scheduled. Good general settings were sought which would work well for the data sets. In practice, the general settings would not need to be changed each time a new set of data is introduced.

The final settings included position-based crossover, a population size of 200, and a selective pressure of 1.5. These settings appear to be robust in giving a good compromise between the best results and execution time.

These parameters were used to produce schedules for all seven days. Experimental runs were performed on Sun Sparc-10 and Sparc-2 workstations. Solution times for one day of data averaged 10 minutes on a Sparc-10 workstation and approximately 24 minutes on a Sparc-2 workstation. On the Sparc-10, producing 1,000 schedules took approximately one minute.

Week One Results. Once a good set of parameters was found, schedules for all seven days of data were produced. The genetic algorithm (GA) results are from GENITOR with a population of 200, selective pressure of 1.5, and up to 8,000 reproductions. Overall results are shown in Figure 12 and Table 4. Supports whose time windows extend into the next day were allowed to be scheduled into the following day. These overlaps were then blocked out of the available RTS time for the next day. This is done by scheduling these overlap requests before the normal support requests in the following day's schedule. The corresponding results from Schalck's mixed-integer programming (MIP) solutions are included for comparison (14:4-2).

The total number of supports requested and scheduled are shown in Table 4. The difference between the MIP and GA-based results are labeled "GA-MIP."

Table 4. Results for Week One Data

Day	# Requested	GA# Scheduled	MIP# Scheduled	GA - MIP
1	322	314	312	2
2	302	296	296	0
3	311	307	304	3
4	318	315	311	4
5	305	301	299	2
6	299	292	292	0
7	297	291	291	0

The genetic algorithm results compare favorably to the MIP results of Schalck. In every case, the number of total supports the genetic algorithm schedules is at least as good as the number of supports scheduled by the MIP.

Breakdown by Support Type. Results are also broken down by low-altitude and high-altitude supports. As mentioned in Chapter I, low-altitude satellite supports are usually given higher priority in scheduling since they are more difficult to reschedule. Here the GA scheduled more high-altitude satellites than in the MIP solution and scheduled fewer low-altitude satellite supports than the MIP approach. This is due to the fitness of a schedule being calculated as the total number of supports scheduled without reference to the support type. Schalck's solution guaranteed scheduling of the greatest number of low-altitude supports possible because they were all scheduled before attempting to schedule any high-altitude satellites. This guarantees the scheduling of the greatest number of low-altitude satellites, but this may not always be necessary. Since set up times for low-altitude supports may be flexible, adjustment of the low-altitude satellite setup times may allow scheduling more supports overall.

Table 5. Low-altitude Results for Week One Data

Day	# Requested	GA# Scheduled	MIP# Scheduled	GA - MIP
1	153	147	149	-2
2	137	132	134	-2
3	146	143	143	0
4	142	139	140	-1
5	142	139	139	0
6	144	138	138	0
7	142	138	138	0

Additional Runs

Since one of the advantages of a hybrid GA approach is flexibility, other constraints on the scheduling of supports should be allowed. The first obvious constraint for inclusion is the scheduling of priorities for different support types.

Table 6. High-altitude Results for Week One Data

Day	# Requested	GA# Scheduled	MIP# Scheduled	GA - MIP
1	169	167	163	4
2	165	164	162	2
3	165	164	161	3
4	176	176	171	5
5	163	162	160	2
6	155	154	154	0
7	155	154	153	1

Support Priority. Low-altitude supports usually take precedence over high-altitude supports since they are more difficult to schedule. To introduce this preference in scheduling, different fitness scores can be assigned to the different types of supports. As an example, low-altitude supports were given a score of two for each one scheduled; the value of high-altitude supports was set to only one. Where a low-altitude support and a high-altitude support compete for the same position in the schedule, the low-altitude support should be scheduled. The overall results are shown in Table 7 and in Figure 13. A breakdown by low-altitude and high-altitude satellites is shown in Table 8 and Table 9.

Table 7. Week One Data with Low-Altitude Supports given Priority

Day	# Requested	GA# Scheduled	MIP# Scheduled	GA - MIP
1	322	313	312	1
2	302	296	296	0
3	311	307	304	3
4	318	314	311	3
5	305	300	299	1
6	299	293	292	1
7	297	291	291	0

Note that more low-altitude supports were accomplished, although at the cost of a tradeoff of some high-altitude supports. Note, however, for Day Two, the GA scheduled two fewer low-altitude supports than the MIP. Upon examination of the schedule, this is due to some high-altitude satellites from Day One scheduled into Day Two.

This type of priority assignment could also be applied to other situations; even so far as to give each satellite a different priority. An array of priority values could be used

Table 8. Low-altitude Results with Low-Altitude Supports given Priority

Day	# Requested	GA# Scheduled	MIP# Scheduled	GA - MIP
1	153	149	149	0
2	137	132	134	-2
3	146	143	143	0
4	142	139	140	-1
5	142	139	139	0
6	144	138	138	0
7	142	138	138	0

Table 9. High-Altitude Results with Low-Altitude Supports given Priority

Day	# Requested	GA# Scheduled	MIP# Scheduled	GA - MIP
1	169	164	163	1
2	165	164	162	2
3	165	164	161	3
4	176	175	171	4
5	163	161	160	1
6	155	155	154	1
7	155	153	153	0

to provide a different score for scheduling each support. A higher score would indicate greater scheduling priority for that support.

Two-Day Scheduling. Even with a priority score given to low-altitude satellite supports, the GA does worse in scheduling low-altitude supports for Days Two and Four. For Day 2, this is caused by the overlap from Day 1. Since each GA only schedules for one day, without regard to the following day, overlaps may reduce the overall number of supports scheduled. The simplest way to overcome this is to schedule two days at once in one block, with priority given to low-altitude supports.

This was attempted with data from Day One and Day Two. Results were encouraging, as the overall number of supports scheduled remains the same, but the number of low-altitude satellite supports scheduled increased to the same level as the MIP solution. In return, the number of high-altitude supports scheduled decreased. Thus, the priority scheme, in conjunction with scheduling for two days, succeeded in matching the low-altitude results of the MIP. Note that the genetic algorithm in this case scheduled two days at once. The solution time was approximately double that for scheduling one day. This indicates that an increase in the number of supports for a day may be handled without much degradation in solution time or quality.

Summary

Results using one week of satellite support data indicates the GA can be successful in scheduling satellite supports. Results match or exceed those returned by the Schalck MIP approach. Solution times are short, and only one computer run is required.

Figure 12. Week One Data Results

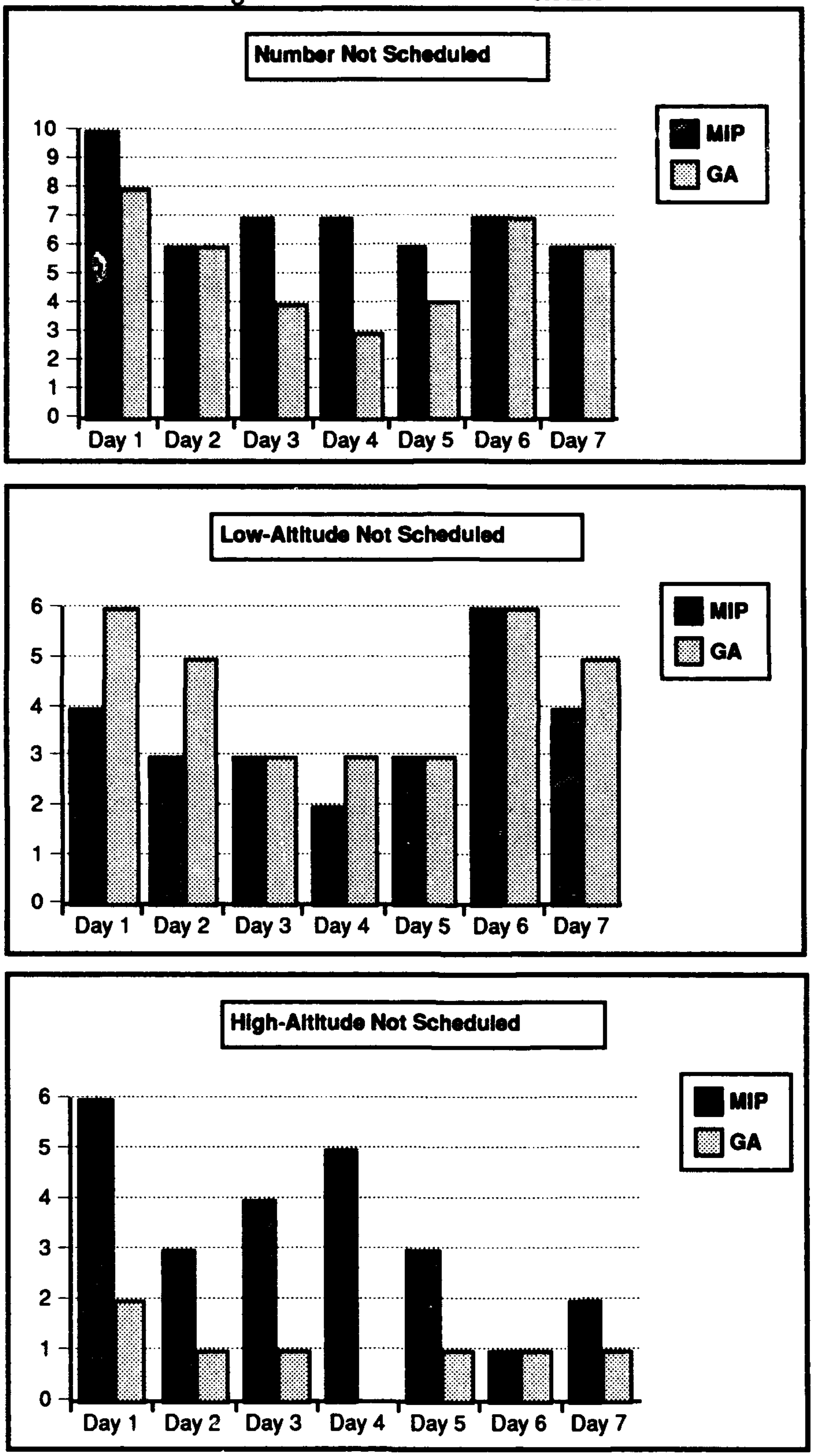
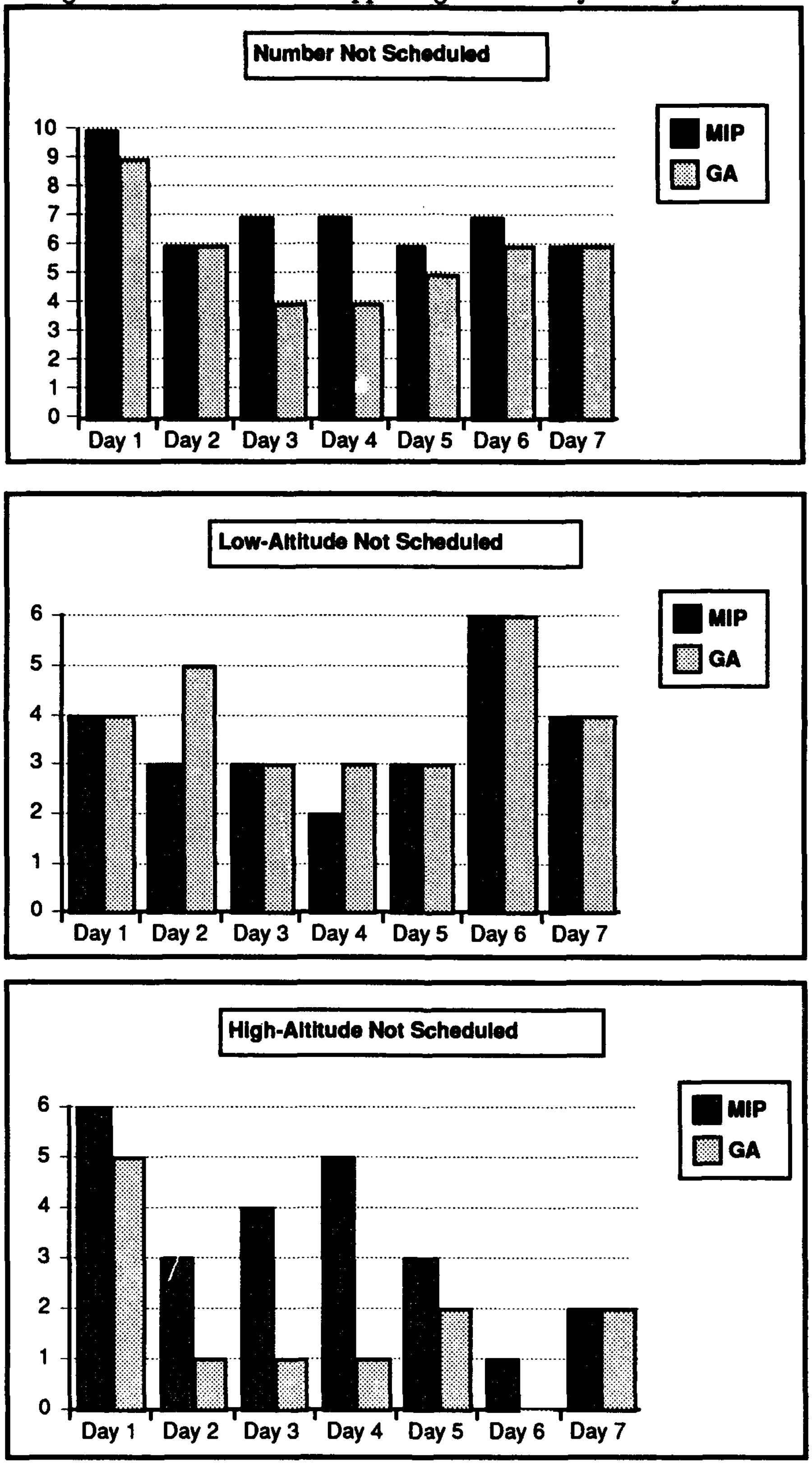


Figure 13. Low-Altitude Supports given Priority for Day 1 Data



V. Conclusions and Recommendations

Conclusions

The satellite range scheduling (SRS) problem involves scheduling over 300 satellite communication supports between Air Force Satellite Control Network (AFSCN) satellites and remote tracking stations (RTS) for a 24-hour period. Scheduling the greatest number of requests minimizes the time needed by schedulers to resolve conflicts in the schedule. The scheduling is currently done manually, with computer aid. Although the current system schedules approximately 95-98% of the requested supports, automating the scheduling process may produce better schedules in less time and use less human resources. The objective of this research was to discover if a genetic algorithm based approach can be effective in automating the scheduling of the requests for a 24-hour period.

The problem can be formulated as a mixed-integer program, but the size of realistic problems precludes producing optimal schedules in a timely manner. By partitioning the original mixed-integer problem into subproblems, Schalck and Gooley found good solutions (14:4-1), but since they do not solve the entire problem at once, there may be room for improvement.

For a test set of seven days of scheduling data, the GA solution matched and often exceeded the results of previous MIP efforts by solving the entire problem at once instead of decomposing the problem into subproblems. The algorithm scheduled over 96% of requested supports. Note, however, these results do not include RTS downtimes. Solution times on Sun Sparc workstations suggest a 24-hour schedule can be produced in under 15 minutes.

These results suggest a genetic algorithm (GA) based method can produce good schedules for the SRS problem. Although the genetic algorithm-based approach described in this research appears to be capable of producing good schedules in a short time, much work remains to be done to implement this automated scheduling approach. Further refinement of the program code may decrease time needed to provide a schedule. More importantly, a fully automated scheduler must also schedule other types of supports which occur in the scheduling process.

Recommendations

Improvements to the algorithm involve further modification and testing of the genetic algorithm with more complicated support requests. The genetic algorithm-based method is easy to implement and modify since all that is required is a schedule builder program to build a schedule one support at a time and to evaluate its fitness score. Since this schedule builder program is easy to test, modifications which allow new constraints can be tested quickly.

Minor Improvements.

Schedule Builder. If the GA-based algorithm is to be implemented, the program code for the schedule builder must be improved. Both readability and code efficiency would lead to better performance and easier modification by users. The program could also be integrated with the processing of satellite request data to make scheduling a one step process for the schedulers, from data processing to final schedule output.

Genetic Algorithm. The genetic algorithm implementation in GENITOR may be improved by adding a mutation operator. Although it does not play as large a role in the genetic search as crossover, mutation acts to keep variety in the population by randomly changing the genes of the population. An order-based mutation operator, such as that advocated by Davis (5:81), would be appropriate for use in this scheduling algorithm. Also, other genetic algorithm packages could be used besides GENITOR. The schedule builder program would simply be used as an evaluation function in the genetic algorithms.

Extensions. To make the produced schedules more realistic, other types of supports should be scheduled. RTS downtimes and scheduling priorities should be determined and included in the schedule. Also, the schedule builder program could be modified to schedule special requests not currently considered by this program. The most important of these would involve scheduling long requests by dividing them into shorter requests which may be scheduled on different RTSs.

Summary. A genetic algorithm based approach offers a way of quickly (minutes) scheduling 24-hour schedules in the satellite range scheduling process. In addition, the general approach could also be applied to other similar scheduling problems where resources must be scheduled. These include ICBM crew scheduling and test range scheduling.

Appendix A. Modified GENITOR Program Code

This appendix describes the GENITOR genetic algorithm, and lists the files modified to use GENITOR with the satellite range scheduling problem. These files include the main program and include files. Although not modified, the source code for the Position crossover operator is included since it is used for the final results.

Genetic Algorithm: GENITOR. The code from an existing genetic algorithm was used for this research. Whitleys GENITOR (GENetic ImplemenTOR) code was chosen because it has been used for similar problems (17). It includes crossover operators for order-based genetic algorithms (edge recombination, order, and PMX). These operators have been shown to be useful in solving many scheduling problems (15).

GENITOR differs from the simple genetic algorithm in two ways. First, instead of creating distinct generations, it creates only one new offspring at a time. Then, rather than replacing the parents with the offspring, GENITOR replaces the lowest ranking member. This approach has been called ***steady-state*** reproduction (5:35). It works by first selecting two parents from the population and producing two offspring. One offspring is randomly discarded. The other offspring replaces the lowest ranking member of the population. This new string is then ranked in relation to the fitness of the other members of the population, and inserted into the population. The new member can then compete for reproductive opportunities.

The second difference from the standard genetic algorithm is the way fitness is measured. Instead of relying on raw fitness measures, GENITOR uses relative ***ranking*** of population members to determine reproductive opportunities. Ranking prevents scaling problems associated with raw fitness values. Scaling problems occur when one individual is so much better that the others that it dominates the population too soon, causing permature convergence of the algorithm. To determine the reproductive opportunities of population members, GENITOR uses linear bias scaling. This linear bias is usually set between 1.0 and 2.0. For example, a bias of 1.5 means the top-ranked individual has 1.5 times the chance of the median-ranked population member to be selected to reproduce (15).

Modified Main program Code. The following listing is the main program code for GENITOR, modified for use with the satellite range scheduling problem. At the beginning of a run, this code reads in satellite time window data and overlap data. During execuation of the program, it calls tsp eval.c to evaluate schedules. Finally, it stores the best sequence in the file tourdata.

This main program is modified for use with the Position crossover operator.

```
/* MAIN Program - 31 Jan 94 */
/*  - 1 Feb, 31 Jan: read/write to files */
/*  - 28 Jan:  add in pre/post processing for overlap  */
/************************************************************/
/*                                                          */
/*  Copyright (c) 1990                                      */
/*  Darrell L. Whitley                                      */
/*  Computer Science Department                             */
/*  Colorado State University                               */
/*                                                          */
/*  Permission is hereby granted to copy all or any part of */
/*  this program for free distribution.  The author's name  */
/*  and this copyright notice must be included in any copy. */
/*                                                          */
/************************************************************/
/***************************************************************************
   main_pos.c
 ***************************************************************************
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "ga_random.h"
#include "gene.h"
#include "ga_global.h"
#include "ga_params.h"
#include "ga_pool.h"
#include "ga_selection.h"
#include "ga_status.h"
#include "ga_signals.h"
#include "op_position.h"

/*******************************************
 Declare evaluation function which tells you
 how "good" a particular gene's solution is.
 NOTE: the input parameters for the eval
       function should always be a gene and
   the gene's length
 ******************************************/
#include "eval_pos.h"

int
main (argc, argv)
int   argc;
char *argv[];
{
 int         i,j;
 GENEPTR     mom, dad, child;
 FILE       *fp;
 int        **coord_array;
 CITY_DATA *city_table;
 /* int          num_diffs; */
/* Character names for input/output */
```

```
 int dayNum;
 static char  overlap[10] =  {"overlap0"};
 static char tourdata[10] = {"tourdata0"};
 static char datafile[10] = {"datafile0"};
/**********************
 Setup signal handlers.
 **********************/
 setup_signal();
/****************************************************************
 Set the global parameters according to command line arguments.
 ****************************************************************/
 argc--;
 argv++;
 parse_command_line (argc, argv);
/**********************
 Print Parameter Values
 **********************/
 fprintf (stdout, "\n");
 print_params(stdout);
 fprintf (stdout, "\n");
/**********************
 Input the day number
 and concat to file names
 **********************/
 printf("What is the day number (1-7)?");
 scanf ("%u", &dayNum);
 printf("Day Number is %u\n",dayNum);
 overlap[7]  += dayNum - 1;     /* Adds day number to filenames */
 tourdata[8] += dayNum;
 datafile[8] += dayNum;
 printf("%s\n%s\n%s\n",tourdata,datafile,overlap);
/********************************
 Seed the Random Number Generator
 ********************************/
 srandom(RandomSeed);
/*******************************************************
 Allocate a genetic pool referenced by the global, Pool
 *******************************************************/
 if ( !(Pool = get_pool(PoolSize, StringLength)) )
fatal_error(NULL);

/******************************************************
 Read in a description of the points to be toured and
 create a representation of the distance between them.
 ******************************************************/
/* make_dist_array (NodeFile, StringLength);  */
/***************************************
  Assign schedule parameters to the program
 *****************************************/
 /* #include "data.h" */
/***************************************
  Read in data
 *****************************************/
  read_data (datafile,Pool->string_length);  /* set = string length */
  read_overlap_data (overlap);               /* 1/28: read overlap data */
 /*  for (i=1; i<=18; i++)
       printf( " gRTS[%d]=%d, gBeginVis[%d]=%d, gEndVis[%d]=%d,
gReqLength[%d]=%d, gTurnAroundTime[%d]=%d \n",i, gRTS[i],i,gBeginVis[i],
 i,gEndVis[i], i, gReqLength[i],i,gTurnAroundTime[i] ); */
 /* printf("Initialize genetic pool\n"); */
/***************************************
 Initialize the genetic pool with data.
 ***************************************/
 init_pool (SeedPool, Pool, 0, Pool->size, tsp_eval);
```

```
/* printf("Sort Initial genetic pool\n"); */
/************************************
 Sort the initial genetic pool data.
 ***********************************/
 sort_pool (Pool);
/* printf("Allocate temporary storage for parents of reproduction\n"); */
/*******************************************************
 Allocate temporary storage for parents of reproduction, and for child.
 ******************************************************/
 mom   = get_gene (Pool->string_length);
 dad   = get_gene (Pool->string_length);
 child = get_gene (Pool->string_length);
/* printf("Allocate a table to be used with the Order1 (Davis) Operator\n"); */
/**************************************************************
 Allocate a table to be used with the Order1 (Davis) Operator
 *************************************************************/
 city_table = get_city_table (Pool->string_length);
/**********
 Optimize !
 **********/
 for (/* CurrentGeneration already set :
     either intialized to 0 in its declaration OR
     initialized by a restart of a previous experiment */;
     CurrentGeneration < NumberTrials;
     CurrentGeneration++)
{
/***********************************
 Choose two genes for reproduction.
 **********************************/
 get_parents(mom, dad, Pool, linear, SelectionBias);
   /*************************************
 Call Order operator to create a child
 *************************************/
     /* printf("Start Order operator\n"); */
if (bitgen() )
{
/* printf("After bitgen = 0\n"); */
position (mom->string, dad->string, child->string, Pool->string_length, city_table);
}
else
{
/* printf("After bitgen = 1\n"); */
position (dad->string, mom->string, child->string, Pool->string_length, city_table);
}
/* printf("So, kid, how good are you?\n"); */
/**************************
 So, kid, how good are you?
 *************************/
     child->worth = tsp_eval (child->string, Pool->string_length);
/* printf("Insert new gene into population according to its worth\n"); */
   /*****************************************************
 Insert new gene into population according to its worth
 *****************************************************/
 insert_gene (child, Pool);
   /**************************************************************
 If the StatusInterval parameter was set and this is the appropriate
 time, print the population best, worst, mean, and average to stdout
 ******************************************************************/
 if (StatusInterval && !(CurrentGeneration % StatusInterval))
    show_progress (stdout, Pool, CurrentGeneration);
    /**************************************************************
 If the DumpInterval parameter was set and this is the appropriate
 time, save the population and key parameters to disk for later
 reference (or to restart execution later.
```

```
******************************************************************/
    if (DumpInterval && !(CurrentGeneration % DumpInterval))
       dump_status(Pool, DumpBase);
}
/*****************
 Summarize Results
 *****************/
 {
  final_pool (FinalPool, Pool, CurrentGeneration);
  fprintf (stdout, "\n");
  printf("Right before print_pool\n");
  print_pool (stdout, Pool, 0, 1);
  printf ("After print_pool, before tourdata\n");
  /****************
 Print out tourdata to file
 *****************/
if (fp = fopen(tourdata, "w"))
{
                printf("Inside print tourdata\n");
  for (i=0;i < Pool->string_length; i++)
{
    fprintf(fp,"%d ",Pool->data[1].string[i]);
}
printf("Its past!\n");
                fclose(fp);
}
else
printf("Cannot print tourdata file");
 /******************************************************************/

 }
}
```

Ga global.h Include files. This file contains the global variables used in GENITOR. The global variables used to store time window information for each support are included here as global variables and used in the tsp eval function. Another related file called ga global external.h is also listed.

```
/*ga_global.h */
/* mod 9 Feb:  add temp LF Num for Priority */
/* modified 8 Feb for downtimes; 28 Jan for overlap */
/**********************************************************/
/*                                                        */
/*  Copyright (c) 1990                                    */
/*  Darrell L. Whitley                                    */
/*  Computer Science Department                           */
/*  Colorado State University                             */
/*                                                        */
/*  Permission is hereby granted to copy all or any part of */
/*  this program for free distribution.   The authors name */
/*  and this copyright notice must be included in any copy. */
/*                                                        */
/**********************************************************/
/*******************************************************************
 * These REQUIRED and OPTIONAL parameters can be chosen by the user. *
 *******************************************************************/

 /* REQUIRED */
 int      PoolSize;
```

```
 int      StringLength;
 int      NumberTrials;
 char     NodeFile[80];       /* contains coordinates of tsp nodes */
 /* OPTIONAL */
 long     RandomSeed;
 float    SelectionBias;
 float    MutateRate = 0.0;
 int      StatusInterval = 0; /* dump status every nth generation */
 int      DumpInterval = 0;   /* save state of every nth generation */
 char     SeedPool[80];       /* file containing init data */
 char     FinalPool[80];      /* file containing final results */
 char     DumpBase[80];       /* basename of file(s) into
                                 which to dump population */
 int      NumPop;             /* Number of Subpopulations */
 int      SwapInterval;       /* Trials between Swapping of Subpopulations */
 int      SwapNumber;         /* Number of Strings Swapped between Subpops */
 int      Experiments;        /* Experiments must be used in main() */
 float    CutOff;             /* Cutoff value for a given experiment */
/****************************************************/
 int SequenceFlag = 0;        /* if set to 1 in main, insert_unique_gene() */
     /* will use different criteria for sameness  */
 int      CurrentGeneration = 0;
 POOLPTR Pool;

/*  Schedule Builder Global Variables */
/*   [spt][win] */
int     gRTS[700][20];
int     gBeginVis[700][20];
int     gEndVis[700][20];
int     gReqLength[700];
int     gTurnAroundTime[700];
int     gBeginSched[700];
int     gEndSched[700];
int     gNumWin[700];
int     gOverlapBegin[20];
int     gOverlapEnd[20];
int gDownNum;
int gDownRTS[100];
int gDownBegin[100];
int gDownEnd[100];
int gNumLF;


/* ga_global_extern.h */
/* mod 9 Feb:  add temp NumLF for priority */
/* mod 8 Feb for downtime */
/* modified 28 Jan 94 to add overlap variables */
/************************************************************/
/*                                                          */
/*  Copyright (c) 1990                                      */
/*  Darrell L. Whitley                                      */
/*  Computer Science Department                             */
/*  Colorado State University                               */
/*                                                          */
/*  Permission is hereby granted to copy all or any part of */
/*  this program for free distribution.  The authors name   */
/*  and this copyright notice must be included in any copy. */
/*                                                          */
```

```
/******************************************************************/
/*****************************************************/
 extern int      PoolSize;
 extern int      StringLength;
 extern int      NumberTrials;
 extern char     NodeFile[];
 extern long     RandomSeed;
 extern float    SelectionBias;
 extern float    MutateRate;
 extern int      StatusInterval;
 extern int      DumpInterval;
 extern char     DumpBase[];
 extern char     SeedPool[];
 extern char     FinalPool[];
 extern int      CurrentGeneration;
 extern int      NumPop;
 extern int      SwapInterval;
 extern int      SwapNumber;
 extern int      Experiments;
 extern float    CutOff;
 extern int      SequenceFlag;
/*****************************************************/
 extern POOLPTR Pool;
/*  Schedule Builder Global Variables */

extern   int     gRTS[700][20];
extern   int     gBeginVis[700][20];
extern   int     gEndVis[700][20];
extern   int     gReqLength[700];
extern   int     gTurnAroundTime[700];
extern   int     gBeginSched[700];
extern   int     gEndSched[700];
extern   int     gNumWin[700];
extern   int     gOverlapBegin[20];
extern   int     gOverlapEnd[20];
extern int gDownNum;
extern int gDownRTS[100];
extern int gDownBegin[100];
extern int gDownEnd[100];
extern int gNumLF;
```

Position Crossover Operator

This program code is unchanged, but included since it was used in GENITOR to test the data.

```
/******************************************************************/
/*                                                                */
/*  Copyright (c) 1990                                            */
/*  Darrell L. Whitley                                            */
/*  Computer Science Department                                   */
/*  Colorado State University                                     */
/*                                                                */
/*  Permission is hereby granted to copy all or any part of       */
/*  this program for free distribution.  The author's name        */
/*  and this copyright notice must be included in any copy.       */
```

```
/*                                                                  */
/******************************************************************/

/******************************************************************
 * *
 * This is the position operator developed by Syswerda *
 *  (The Genetic Algorithms Handbook, L Davis, ed) and  *
 * implemented by Susan McDaniel at Colorado State  *
 *  University.      *
 * *
 * *
 * To use this program include the following lines in  *
 * the main_tsp.c file: *
 * *
 * *
 * This call should only happen once: *
 * get_city_table (Pool->string_length); *
 * *
 * *
 * This code should be in a loop so it is called once *
 * for each recombination: *
 * if ( bitgen () ) *
 *    position (mom->string, dad->string, child->string,  *
 *               Pool->string_length, city_table); *
 *      else *
 *    position (dad->string, mom->string, child->string,  *
 *    Pool->string_length, city_table); *
 * *
 * *
 * *
 * *
 * This operator attempts to preserve position  *
 * information during the recombination process.  *
 * Several random locations in the tour are selected  *
 * along with one of the parents and the cities in  *
 * those positions are inherited from that parent.  *
 * The remaining cities are inherited in the order in  *
 * which they appear in the unselected parent skipping  *
 * over all cities which have already been included in  *
 * the offspring. *
 * *
 * Example- Position based crossover: *
 * Parent 1:  a b c d e f g h i j *
 * Cross Pts:    * *      *      *  (Parent 1 selected) *
 * Parent 2: c f a j h d i g b e *
 * *
 * Offspring: a b c j h f d g i e *
 * *
 * The cities b, c , f and i are inherited from  *
 * Parent 1 in positions 2, 3, 6 and 9 respectively.  *
 * The remaining cities are inherited from Parent 2 as  *
 * follows:  Off[1] = P2[3] since P2[1] and P2[2] have  *
 * already been included in the child tour.  Then  *
 * going through Parent 2 in order, Off[4] = P2[4],  *
 * Off[5] = P2[5], Off[7] = P2[6], Off[8] = P2[8] and  *
 * Off[10] = P2[10]. *
 * *
 ******************************************************************/

#include <stdio.h>
#include <malloc.h>
#include "ga_random.h"
#include "gene.h"
#include "op_position.h"


/*******************************************************************
 *  FUNCTION: get_city_table
 *
 *  DESCRIPTION: allocates space for the city data table
 *  the city data table contains the position
 *  of each city in each parent and also
 *  has a field which is set when a city has
```

```
 *  been included in the offspring tour.
 *
 * INPUT PARAMETERS: string length
 *
 *
 * RETURN VALUE: the address of the city table
 *
 *****************************************************************/
CITY_DATA *
get_city_table (length)
int length;
{
    CITY_DATA *city_table;

    /*  malloc one extra location so that cities 1-N can be accessed
directly.  location 0 will not be used  */

    if (!(city_table = (CITY_DATA *) malloc ((length + 1)*sizeof (CITY_DATA))))
       printf ("get_city_table: Malloc failure. \n");

    return (city_table);
}


/*****************************************************************
 *  FUNCTION: position
 *
 *  DESCRIPTION: performs position crossover operation
 *
 * INPUT PARAMETERS: two parent gene strings
 *      space for child string
 *      the length of the two strings
 *      the city table address
 *
 *
 * RETURN VALUE:
 *
 *****************************************************************/
void position (dad, mom, kid, length, city_table)
GENE_DATA dad[], mom[], kid[];
int length;
CITY_DATA *city_table;
{
    int num_positions;
    int i, pos, dad_index, kid_index;

    for (i=1; i<=length; i++) { /* initialize city table */
       city_table[i].used = 0;
    }
    /* select #positions that will be inherited directly from parent */
    num_positions = randomain (2*length/3, length/3);
    for (i=0; i<num_positions; i++)   {
       pos = randomain (length - 1, 0); /* select position randomly */
       kid[pos] = mom[pos]; /* transfer cities to child */
       city_table[mom[pos]].used = 1; /* mark city used */
    }

    dad_index = 0;
    kid_index = 0;
    while (kid_index < length) {
       if (!city_table[mom[kid_index]].used) { /* next position in kid filled*/
  if (!city_table[dad[dad_index]].used) {  /*next city in dad not used*/
     kid[kid_index] = dad[dad_index]; /* inherit from dad */
     dad_index ++; /* increment indexes */
     kid_index ++;
  }
```

```
else  { /* next city in dad has been used */
   dad_index ++; /* so increment dad index */
  }  /* end else */
      }   /* end if */
      else  { /* next position in kid is filled */
kid_index ++; /* so increment kid index */
      }   /* end else */
  }   /* end while */
 }
}
```

Appendix B. Evaluation Function Code

The tsp eval program is a procedure which builds and evaluates a schedule. It takes as input an ordered list of supports to schedule and then, in the order of the list, attempts to schedule each support into a schedule. The function returns a fitness value to the main genetic algorithm.

The schedule builder code includes three procedures: ScheduleWindow, Schedule-Support, and tsp eval. Tsp eval is the main procedure. It is called from GENITOR, and returns the evaluation of a schedule. Tsp eval first clears a schedule, schedules any overlaps from the previous day, then attempts to schedule each support in the order given in the sequence passed to it from GENITOR. ScheduleSupport attempts to schedule each support in the available time windows. Each time window is attempted by calling the ScheduleWindow routine. If a support is scheduled in a window, the space taken by the support is blocked out of the schedule. Then ScheduleSupport returns a fitness value to tsp eval. This fitness value is normally one, but can differ if a priority scheme is used to weight the scheduling of different types of supports. After all supports have been scheduled (or failed to be scheduled), tsp eval returns a fitness score based on the number of supports scheduled to the main GENITOR function.

```
/* SCHEDULE BUILDER PROGRAM */
/* 31 Jan 94:  auto file generation */
/* 28 Jan 94 Eval_tsp.c -- position crossover */
/* 28 Jan:  add overlap variables */
/* This modification allows supports to be scheduled past the end of the day */
/* if allowed by the supports windows */
/*  do this by increasing filled array from 1470 to 2100, and take off end */
/*  conditions                                                              */
#include <stdio.h>
#include <math.h>
#include "gene.h"
#include "op_position.h"
#include "ga_global_extern.h"
/*****************************************************************/
int
ScheduleWindow (spt, win, filled)
int   spt;
int   win;
int filled[21][2100];
{
     /* Declare Variables */
     int
          bVis,           /* gBeginVis - gTurnAroundTime : 0 if latter <0 */
    eVis,       /* ending visibility */
          rLength,      /* RegLength + TurnAroundTime */
         endSpt,     /* Flag for Support either scheduled or run out */
```

```
        j,              /* Counter for support index being scheduled */
        l,              /* Counter for filling in scheduled blocks */
                /*  spt:            Index for support */
        minute,         /* current time interval */
        length,     /* length so far */
        scheduled,  /* binary: 1 or scheduled; 0 for not schedule */
        beginSched,
        endSched,
        start,      /* current start of support */
 tat,           /* turn around time */
        rts;        /* RTS number -- passed from global gRTS[support] */
                     /* scheduledPtr : indicates if scheduled or not */
    /**** Initialize Variables *****/
    scheduled = 0;
    length = 0;
    endSpt =0;  /* initialize flag; end when it = 1*/

    /* Initialize Starting Point, Required Length with TAT */
    rts = gRTS[spt][win];
    tat = gTurnAroundTime[spt];
    bVis = gBeginVis[spt][win] - gTurnAroundTime[spt] + 20 ;
    eVis = gEndVis[spt][win] + 20;
/* add twenty for overlap area */
    if (bVis < 0)
{
          tat = bVis + tat;
   bVis = 0;        /**** reduces tat if near boundary at start ****/
}
    rLength = gReqLength[spt] + tat;
    minute= bVis + 1;    /* my blocks are filled to the right of position */
    start = minute;             /* Starting position of current attempt */
    while  (endSpt != 1)     /* schedule until successful or end */
    {
         if (filled[rts][minute] == 0)    /* if current space free */
              length += 1;              /* increment length if space free */
         else
         {
              length = 0;  /* reset to zero if space is filled */
    tat = gTurnAroundTime[spt];
        rLength = gReqLength[spt] + tat;
              start = minute+1;
         }
         if (length == rLength)   /* then schedule the support here */
              {
                   beginSched = minute - gReqLength[spt] - 20;
   endSched = minute - 20;
                   scheduled = 1;
                   for (l = start;l <= minute;l++)
                      filled[rts][l] = 1;
   }
         if ( (minute == eVis) || (scheduled == 1) )
            endSpt = 1;
         minute += 1;
    } /* End Support */
     /*  fprintf (stdout, "scheduled = %d\n", scheduled); */
     /*if(scheduled == 1)
       printf("SCHEDULE Support %d  in window %d at RTS %d, begin: %d
   end: %d\n",
                     spt, win, rts, beginSched, endSched );
    else
       printf("     Support %d not scheduled in window %d\n", spt, win); */
```

```
        return(scheduled);
} /* end ScheduleWindow */
```

```
/********************************************************************/
int
ScheduleSupport(spt,filled)
int spt;
int filled[21][2100];
{
int i, win, flag;
int sched;    /* flag = 1 if scheduled , 0 ow */
sched = 0;
win = 1;            /* counter for number of windows tried */
flag = 0;
while (flag != 1)
  {
     sched = ScheduleWindow(spt,win,filled);
    /*  printf ("sched=%d, gNumWin[%d]=%d\n", sched,spt,gNumWin[spt]); */
     if ( (sched == 1) || (win == gNumWin[spt]) )
flag =1;
     win +=1;
}
/* if (sched==0)
  printf("***  Support %d NOT SCHEDULED!***\n",spt); */
return(sched);
}

/* ************************************************************** */
float
tsp_eval (order, num_supports)
GENE_DATA order[];
int       num_supports;
{
     /* Declare Variables */
     int
         filled[21][2100],   /* time units in schedule are filled[rts][time] */
         j,               /* Counter for support index being scheduled */
         k,
         spt,             /* Index for support */
         numScheduled,    /* number of supports scheduled */
          minute,          /* counter for current time */
         rtsNum;          /* max number of RTSs */
     /**** Initialize Variables *****/
    rtsNum = 20;  /* must be one less than max cause of arrays 0-19 */
     /* printf("rtsNum = %d\n", rtsNum); */
    for (k = 1;k <= rtsNum;k++)
    {
        for (minute = 0;minute <= 2100;minute++)
        {
            filled[k][minute] = 0;
           /* printf("filled[%d][%d]= %d\n",k,minute,filled[k][minute]); */
        }
     }
     /* Schedule the overlaps first, before the real supports */
     for (k = 1; k <= rtsNum; k++)
  for (minute = gOverlapBegin[k]; minute <= gOverlapEnd[k]; minute++)
     filled[k][minute] = 1;

     numScheduled = 0;

     /* Schedule each Support in Priority Order */
     for (j = 0;j< num_supports; j++)    /* Try to schedule support */
     {
          spt = order[j];
```

```
            /*  printf( "Start SS Order[%d] = %d\n",j, order[j]);  */
            numScheduled += ScheduleSupport(spt, filled);
     }      /* end num_supports */
    /* fprintf (stdout, "num_supports = %d\n", num_supports); */
   /* The GA wants to MINimize something... so do Max poss - actual */
      /* printf( "\n Number Scheduled = %d\n**************************\n\n\n",
 numScheduled); */
     return(num_supports - numScheduled);
}     /* End Eval */


/****************************************************************************
 FUNCTION: read_data
 *
 * DESCRIBE: read in data
 *
 * INPUT PARAMETERS: filename of list of satellite support data
 *
 *
 *
 * RETURN VALUE: none
 *
 * CALLS: get_coord_array
 *        read_coords
 *        get_2D_dist_array
 *        calc_distances
 ****************************************************************************/
void
read_data (coord_file, num_supports)
char            coord_file[];
int             num_supports;
{
  int i;
  int spt;
  int rts;
  int bVis, eVis, rLength,tat;
  FILE *fp;
  printf("coord_file is: %s", coord_file);
  for (i=1; i<=num_supports; i++)
{
gNumWin[i] = 0;      /* set number of windows [spt] = 0 */
/* printf("gNumWin[%d]=%d\n",i,gNumWin[i]); */
}
  if (fp = fopen(coord_file, "r"))
{
         while (fscanf(fp, "%d %d %d %d %d %d", &spt, &rts, &bVis,
 &eVis,&rLength, &tat ) !=EOF)
         {
                gNumWin[spt] +=1;
                gRTS[spt][gNumWin[spt]] = rts;
                gBeginVis[spt][gNumWin[spt]] = bVis; /* add for overlap */
                gEndVis[spt][gNumWin[spt]] = eVis;   /* add for overlap */
                gReqLength[spt] = rLength;
                gTurnAroundTime[spt] = tat;
                /* printf ("spt = %d, rts= %d, Bvis= %d, Evis = %d, ReqLen = %d,
 TAT = %d\n", spt, gRTS[spt][gNumWin[spt]],  gBeginVis[spt][gNumWin[spt]],
 gEndVis[spt][gNumWin[spt]],  gReqLength[spt],  gTurnAroundTime[spt]  ); */
      /* printf("------gNumWin[%d] = %d\n", spt,gNumWin[spt]); */
         }
 fclose (fp);
  /* for (i=1; i<= num_supports;i++)
printf("gNumWin[%d] = %d\n",i,gNumWin[i]); */
}
 else
```

```
fatal_error ("Cannot read input city file datafile");
}


/*****************************************************************************
 FUNCTION: read_overlap_data
 *
 * DESCRIBE: read in data
 *
 * INPUT PARAMETERS: filename of list of satellite support overlap data
 *
 *
 *
 * RETURN VALUE: none
 *
 * CALLS: get_coord_array
 *        read_coords
 *        get_2D_dist_array
 *        calc_distances
 *****************************************************************************/
void
read_overlap_data (coord_file)
char            coord_file[];
{
  int i;
  int rts;
  int bVis, eVis;
  FILE *fp;

  if (fp = fopen(coord_file, "r"))
{
        while (fscanf(fp, "%d %d %d", &rts, &bVis, &eVis ) !=EOF)
        {
              gOverlapBegin[rts] = bVis;
      gOverlapEnd[rts] = eVis;
              /* printf("gOverlapBegin[%d] = %d\t gOverlapEnd[%d] = %d\n", rts,
 gOverlapBegin[rts], rts, gOverlapEnd[rts]); */
              /* printf ("spt = %d, rts= %d, Bvis= %d, Evis = %d, ReqLen = %d,
 TAT = %d\n", spt, gRTS[spt][gNumWin[spt]],  gBeginVis[spt][gNumWin[spt]],
 gEndVis[spt][gNumWin[spt]],  gReqLength[spt],  gTurnAroundTime[spt]  ); */
      /* printf("------gNumWin[%d] = %d\n", spt,gNumWin[spt]); */
        }
 fclose (fp);
  /* for (i=1; i<= num_supports;i++)
printf("gNumWin[%d] = %d\n",i,gNumWin[i]); */
}
 else
fatal_error ("Cannot read input city file overlap");
}
```

Appendix C. Schedule Builder

This program code is a modification of the Evaluation Function code in Appendix B. Schedule Builder generates a schedule corresponding to an input solution sequence. This solution is stored in a file named ***tourdata*** and is produced by the GENITOR code at the end of a run.

The output of the Schedule Builder code is an overlap file and a schedule file. The ***overlap*** file contains the start and end times of all supports which overlap into the next day or within the maximum turn around time of the end of the day. The ***schedule*** file contains the schedule produced for the 24-hour period. Included in this file is the support number, RTS, beginning time, ending time, service time, and turnaround time.

```
/* Schedule Builder Code - build schedule */
/* 1 Feb:  print schedule to file */
/* 30 Jan 94:  fix overlap by clearing the global overlaps */
/* 28 Jan 94:  overlap of 20 minutes allowed at beginning */
/* 3 Jan 94:  allowed supports to be scheduled past 1470 */
/* changed filled[] array from 1470 to 2100, take off condition */
#include <stdio.h>
#include <math.h>
#include <gene.h>
#include <op_edge_recomb.h>
#include <ga_global_extern.h>
/***************************************************************/
int
ScheduleWindow (spt, win, filled)
int   spt;
int   win;
int filled[21][2100];
{
     /* Declare Variables */
     int
          bVis,              /* gBeginV.    gTurnAroundTime : 0 if latter <0 */
          eVis,              /* ending visibility 28 Jan */
          rLength,        /* RegLength + TurnAroundTime */
         endSpt,      /* Flag for Support either scheduled or run out */
         j,                /* Counter for support index being scheduled */
         l,                /* Counter for filling in scheduled blocks */
                /*  spt:              Index for support */
         minute,           /* current time interval */
         length,      /* length so far */
         scheduled,  /* binary: 1 or scheduled; 0 for not schedule */
         beginSched,
         endSched,
         start,       /* current start of support */
 tat,         /* turn around time */
         rts;         /* RTS number -- passed from global gRTS[support] */
                        /* scheduledPtr : indicates if scheduled or not */
FILE *fp;
     /**** Initialize Variables *****/
     scheduled = 0;
     length = 0;
```

```
     endSpt =0;  /* initialize flag; end when it = 1*/
      /* Open file for output to schedule  1 Feb */
fp = fopen(  schedule  ,   a  );

     /* Initialize Starting Point, Required Length with TAT */
     rts = gRTS[spt][win];
     tat = gTurnAroundTime[spt];
     bVis = gBeginVis[spt][win] - gTurnAroundTime[spt] + 20;
     eVis = gEndVis[spt][win] + 20;   /* 20 is max TAT  - 28 Jan */
/* if ( (spt==1) || (spt==155) )
printf(  tat=%d,bVis=%d\n  ,tat,bVis); */
     if (bVis < 0)
{
           tat = bVis + tat;
   bVis = 0;       /**** reduces tat if near boundary at start ****/
}
     rLength = gReqLength[spt] + tat;
/* if ( (spt==1) || (spt==155) )
printf(  rLength=%d\n  ,rLength); */
     minute= bVis + 1;    /* my blocks are filled to the right of position */
     start = minute;             /* Starting position of current attempt */
     while  (endSpt != 1)     /* schedule until successful or end */
     {
          if (filled[rts][minute] == 0)     /* if current space free */
               length += 1;               /* increment length if space free */
          else
          {
               length = 0;  /* reset to zero if space is filled */
   tat = gTurnAroundTime[spt];
         rLength = gReqLength[spt] + tat;
               start = minute+1;
          }
          if (length == rLength)
               {
                    beginSched = minute - gReqLength[spt] - 20;
           endSched = minute - 20;
                    scheduled = 1;
                    for (l = start;l <= minute;l++)
                       filled[rts][l] = 1;

                    /* If ends in TAT zone then save to overlap */
           if (endSched > 1420)       /* 1440 - max TAT; in overlap */
              if ( (gOverlapEnd[rts] = 0) || (endSched >
 gOverlapEnd[rts]) )
        {  /* if rts overlap is new or this one is longer than
 last */
     gOverlapEnd[rts] = endSched - 1420;
     if (beginSched > 1420 )
          gOverlapBegin[rts] = beginSched - 1420;
     else
          gOverlapBegin[rts] = 0;
        }
               }
          if ( (minute== eVis) || (scheduled == 1) )
             endSpt = 1;
          minute += 1;
     } /* End Support */
      /*  fprintf (stdout,   scheduled = %d\n   , scheduled); */
      if(scheduled == 1)
{
         printf(  %d %d %d %d %d %d\n   ,
                     spt, rts, beginSched, endSched , gReqLength[spt],
```

```
 gTurnAroundTime[spt] );
fprintf(fp,  %d %d %d %d %d %d\n  ,
                          spt, rts, beginSched, endSched , gReqLength[spt],
 gTurnAroundTime[spt] );
}
      /*else
        printf(       Support %d not scheduled in window %d\n  , spt, win); */
     fclose(fp);  /* stop printing to file 1 Feb */
     return(scheduled);
} /* end ScheduleWindow */
/***************************************************************/
int
ScheduleSupport(spt,filled)
int spt;
int filled[21][2100];
{
int i, win, flag;
int sched;    /* flag = 1 if scheduled , 0 ow */
sched = 0;
win = 1;             /* counter for number of windows tried */
flag = 0;
while (flag != 1)
  {
     sched = ScheduleWindow(spt,win,filled);
    /*  printf (  sched=%d, gNumWin[%d]=%d\n  , sched,spt,gNumWin[spt]); */
     if ( (sched == 1) || (win == gNumWin[spt]) )
flag =1;
     win +=1;
}
/* if (sched==0)
  printf(  ***  Support %d NOT SCHEDULED!***\n  ,spt);  */
return(sched);
}

/* ************************************************************ */
float
tsp_eval (order, num_supports)
int order[];
int       num_supports;
{
     /* Declare Variables */
     int
        filled[21][2100],   /* time units in schedule are filled[rts][time] */
        j,               /* Counter for support index being scheduled */
        k,
        spt,           /* Index for support */
        numScheduled,    /* number of supports scheduled */
         minute,          /* counter for current time */
        rtsNum;          /* max number of RTSs */
     /**** Initialize Variables *****/
    rtsNum = 20;  /* must be one less than max cause of arrays 0-19 */
     /* printf(  rtsNum = %d\n  , rtsNum); */
    for (k = 1;k <= rtsNum;k++)
    {
        for (minute = 0;minute <= 2100;minute++)
        {
           filled[k][minute] = 0;
          /* printf(  filled[%d][%d]= %d\n  ,k,minute,filled[k][minute]); */
        }
     }
   /* Schedule the overlaps first, before the real supports */
```

```
    for (k = 1; k <= rtsNum; k++)
   for (minute = gOverlapBegin[k]; minute <= gOverlapEnd[k]; minute++)
filled[k][minute] = 1;
   /* Set the Overlaps back to zero; since written to in scheduling 30 Jan */
      for (k = 1; k <= rtsNum; k++)
      {
   gOverlapBegin[k] = 0;
          gOverlapEnd[k] = 0;
      }
    numScheduled = 0;
    /* Schedule each Support in Priority Order */
printf(  Spt RTS Beg End Length TAT\n\n  );
    for (j = 1;j<= num_supports; j++)    /* start at 0 for Tsp, 1 for sb */
    {
         spt = order[j];
            /* printf(   Start SS Order[%d] = %d\n  ,j, order[j]); */
         numScheduled += ScheduleSupport(spt, filled);
    }     /* end num_supports */
   /* fprintf (stdout,   num_supports = %d\n  , num_supports); */
  /* The GA wants to MINimize something so do Max poss - actual */
     printf(   \n Number Scheduled = %d\n***************************\n\n\n  ,
 numScheduled);
     return(num_supports - numScheduled);
}    /* End Eval */


/*****************************************************************************
 FUNCTION: read_data
 *
 * DESCRIBE: read in data
 *
 * INPUT PARAMETERS: filename of list of satellite support data
 *
 *
 *
 * RETURN VALUE: none
 *
 * CALLS: get_coord_array
 *        read_coords
 *        get_2D_dist_array
 *        calc_distances
 *****************************************************************************/
void
read_data (coord_file, num_supports)
char             coord_file[];
int             num_supports;
{
  int i;
  int spt;
  int rts;
  int bVis, eVis, rLength,tat;
  FILE *fp;
  for (i=1; i<=num_supports; i++)
{
gNumWin[i] = 0;      /* set number of windows [spt] = 0 */
/* printf(  gNumWin[%d]=%d\n  ,i,gNumWin[i]); */
}
  if (fp = fopen(coord_file,   r  ))
{
        while (fscanf(fp,   %d %d %d %d %d %d  , &spt, &rts, &bVis, &eVis,&rLength, &tat ) !=EOF)
        {
             gNumWin[spt] +=1;
             gRTS[spt][gNumWin[spt]] = rts;
             gBeginVis[spt][gNumWin[spt]] = bVis;
```

```
                gEndVis[spt][gNumWin[spt]] = eVis;
                gReqLength[spt] = rLength;
                gTurnAroundTime[spt] = tat;
                /* printf (  spt = %d, rts= %d, Bvis= %d, Evis = %d, ReqLen = %d,
 TAT = %d\n  , spt, gRTS[spt][gNumWin[spt]],  gBeginVis[spt][gNumWin[spt]],
 gEndVis[spt][gNumWin[spt]],  gReqLength[spt],  gTurnAroundTime[spt]  ); */
          }
 fclose (fp);
 /* for (i=1; i<= num_supports;i++)
printf(  gNumWin[%d] = %d\n  ,i,gNumWin[i]); */
}
 else
fatal_error (  Cannot read input city file  );
}

/******************************************************************************
 FUNCTION: read_overlap_data
 *
 * DESCRIBE: read in data
 *
 * INPUT PARAMETERS: filename of list of satellite support overlap data
 *
 *
 *
 * RETURN VALUE: none
 *
 * CALLS: get_coord_array
 *        read_coords
 *        get_2D_dist_array
 *        calc_distances
 ******************************************************************************/
void
read_overlap_data (coord_file)
char            coord_file[];
{
  int i;
  int rts;
  int bVis, eVis;
  FILE *fp;

  if (fp = fopen(coord_file,   r  ))
{
        while (fscanf(fp,   %d %d %d  , &rts, &bVis, &eVis ) !=EOF)
        {
             gOverlapBegin[rts] = bVis;
      gOverlapEnd[rts] = eVis;
              /* printf(  gOverlapBegin[%d] = %d\t gOverlapEnd[%d] = %d\n  ,
 rts, gOverlapBegin[rts], rts, gOverlapEnd[rts]); */
             /* printf (  spt = %d, rts= %d, Bvis= %d, Evis = %d, ReqLen = %d,
 TAT = %d\n  , spt, gRTS[spt][gNumWin[spt]],  gBeginVis[spt][gNumWin[spt]],
 gEndVis[spt][gNumWin[spt]],  gReqLength[spt],  gTurnAroundTime[spt]  ); */
      /* printf(  ------gNumWin[%d] = %d\n  , spt,gNumWin[spt]); */
          }
 fclose (fp);
  /* for (i=1; i<= num_supports;i++)
printf(  gNumWin[%d] = %d\n  ,i,gNumWin[i]); */
}
 else
fatal_error (  Cannot read input city file  );
}

/******************************************************************************
 FUNCTION: print_overlap_data
 *
 * DESCRIBE: print out data
 *
```

```
 * INPUT PARAMETERS: filename of list of satellite support overlap data
 *
 *
 *
 * RETURN VALUE: none
 *
 ******************************************************************************/
void
print_overlap_data (coord_file)
char            coord_file[];
{
  int i;
  int rts;
  int bVis, eVis;
  FILE *fp;

  if (fp = fopen(coord_file,   w  ))
{
      for (rts = 1; rts <= 19; rts++)
      {
                  bVis = gOverlapBegin[rts];
          eVis = gOverlapEnd[rts];
            fprintf(fp,   %d\t%d\t%d\n  , rts, bVis, eVis );
              }

      fclose (fp);
}
 else
fatal_error (  Cannot read or write overlap file  );
}
```

Appendix D. Satellite Range Scheduling Data Processing

The processing of satellite support data begins with raw ASTRO data which is processed by Pascal programs to produce satellite request windows. This data is filtered by a C program to prepare the data for the genetic algorithm.

ASTRO Data

ASTRO (Automated Scheduling Tools for Range Operations) is a computer system and database to aid the range schedulers. The report information for the seven days of test data is in a file named FINLDATA.DFT. This file is the raw input which determines support requests and request visibilities.

Requests and Visibility Windows

Pascal programs written by Gooley and Schalck are used to process the ASTRO data. The programs extract the satellite support requirements and request visibilities. Five programs are used for this algorithm: one to process the low-altitude satellite (low-flyer) support information, and four to process the medium and high-altitude satellite support (high-flyer) information. These programs are run using TurboPascal on a IBM PC compatible computer.

The final data format is:

1. Support Number
2. RTS the satellite is visible to
3. Beginning visibility time of window (in minutes)
4. Ending visibility time of window
5. Length of support (in minutes)
6. TAT (turn around time) required by an RTS
7. IRON/Revolution: identifies satellite and pass

LREQ.PAS. This Pascal program reads in the information on low-flyer supports and saves the information to a file called REQLF.DAT containing the low altitude satellite support requests for a day.

HREQ.PAS. This program reads in the information on high-altitude requests for a day and saves the information to two files: 1) REQHF.DAT: the high-altitude requests for a day, and 2) D1V.DAT: the visibilities for the high altitude requests for a day.

TOL.PAS. This program takes tolerance data presented in different formats for each request from HREQ.PAS and standardizes the tolerance window data.

CROSS2.PAS. This program cross checks requests and visibilities. It cross references the visibility file created in HREQ.PAS and the output from TOL.PAS to determine all the RTSs that can satisfy each medium or high altitude support request from TOL.PAS.

RTS.PAS. This Pascal program ensures all RTS sides are included once and only once for each support request-RTS visibility combination.

Prepare Time Window Data for GENITOR

This program reads in the separate data files for the low-flyers and high-flyers and combines them into one file for use in the GENITOR genetic algorithm. It also changes the RTS name to a number for reference in array structures.

```
/* 5 Jan 94 Preprocess Data  modications */
/* now strips out all windows for RTS # 9, 16, 19  to match Spike */
/*   --  reads in request windows of LF/HF and prints out concate file */
/*       with RTS names changed to numbers                             */
/*  1) Open output file "Dayout.dat"  */
/*  2) Ask for LF file name ( ex. "LFDay1g.dat") */
/*  3) Read in data, convert name to number and drop IRON */
/*  4) Ask for HF file name, read in data -- start spt # at LF +1 */
/**************************************************************************
 Program: Process.
 *
 * DESCRIBE: read in data
 *
 * INPUT PARAMETERS: filename of list of satellite support data for LF and HF
 *
 *
 *
 * RETURN VALUE: none, but print a support list to Dayout.dat
 *
 * CALLS: read_data
```

```
 ***************************************************************************/
#include <stdio.h>

int
equal_strings (s1, s2)
char s1[],s2[];
{
int i = 0, answer;
while ( s1[i] == s2[i] && s1[i] != '\0' && s2[i] != '\0' ) ++i;
if ( s1[i] == '\0' && s2[i] == '\0' )
answer = 1; /* strings equal */
else
answer = 0; /* not equal     */
return (answer);
}
int
name_to_Num (name)
char name[];
{
int rts;
if      (equal_strings (name,"POGO-A") )
rts = 1;
else if (equal_strings (name,"POGO-B") )
rts = 2;
else if (equal_strings (name,"POGO-C") )
rts = 3;
else if (equal_strings (name,"HULA-A") )
rts = 4;
else if (equal_strings (name,"HULA-B") )
rts = 5;
else if (equal_strings (name,"COOK-A") )
rts = 6;
else if (equal_strings (name,"COOK-B") )
rts = 7;
else if (equal_strings (name,"INDI-A") )
rts = 8;
else if (equal_strings (name,"INDI-B") )
rts = 0;
else if (equal_strings (name,"BOSS-A") )
rts = 10;
else if (equal_strings (name,"BOSS-B") )
rts = 11;
else if (equal_strings (name,"LION-A") )
rts = 12;
else if (equal_strings (name,"LION-B") )
rts = 13;
else if (equal_strings (name,"GUAM-A") )
rts = 14;
else if (equal_strings (name,"GUAM-B") )
rts = 15;
else if (equal_strings (name,"PIKE-A") )
rts = 16;
else if (equal_strings (name,"PIKE-B") )
rts = 0;
else if (equal_strings (name,"REEF-A") )
rts = 18;
else if (equal_strings (name,"REEF-B") )
rts = 0;
else
printf("UNKNOWN\n");
return(rts);
}
read_data (out_file, coord_file, start, endPtr)
FILE *out_file;
```

```
char        coord_file[];
int start;
int  *endPtr;
{
int i;
   int spt;
   char rtsName[8];
   int bVis, eVis, rLength,tat;
   char iron[16];
   int totNumSpt;
   int rts;
   FILE *input_file;

   input_file = fopen(coord_file, "r");
   while (fscanf(input_file, "%d %s %d %d %d %d %s", &spt, &rtsName, &bVis,
 &eVis,&rLength, &tat, &iron ) !=EOF)
    {
 spt = spt + start;
 totNumSpt = spt;
        rts = name_to_Num(rtsName);
 if ( rts != 0 )
           fprintf (out_file,"%d\t%d\t%d\t%d\t%d\t%d\n", spt, rts, bVis, eVis,
 rLength, tat);
    }
   fclose (input_file);
   *endPtr = totNumSpt;
}

/*****    MAIN PROGRAM  *****/

main()
{  /* start MAIN */
/**** Initialize and declare Variables *****/
char lfdata[12], hfdata[12];
int NumLFSpt, NumHFSpt;
FILE  *output_file;

output_file = fopen("datafile", "w");

/**** Input LF Filename *****/
printf("What is the LF filename?");
scanf ("%s", lfdata);

/**** Read in LF Data and print to Dayout.dat *****/
read_data (output_file, lfdata, 0, &NumLFSpt);

/**** Input HF filename *****/
printf("What is the HF filename?");
scanf ("%s", hfdata);

/**** Read in HF Data and print to Dayout.dat *****/
read_data (output_file, hfdata,NumLFSpt, &NumHFSpt);

fclose (output_file);
printf("NumLFSpt = %d, EndHFSpt = %d\n", NumLFSpt, NumHFSpt);
} /* end Main */
```

Appendix E. Schedules for Week One Data

Schedules produced for each of seven days of data follow. The scheduled supports are sorted by RTS, and by time. The columns are: Support number, RTS, Beginning Time, Ending Time, Support Length, and Support turn around time.

Day 1

2 POGO-A 26 42 16 20
167 POGO-A 65 80 15 15
170 POGO-A 95 105 10 15
10 POGO-A 138 152 14 20
180 POGO-A 180 185 5 15
22 POGO-A 234 248 14 20
27 POGO-A 268 279 11 20
193 POGO-A 294 304 10 15
31 POGO-A 331 345 14 20
39 POGO-A 367 379 12 20
204 POGO-A 395 410 15 15
205 POGO-A 425 435 10 15
47 POGO-A 466 480 14 20
51 POGO-A 527 543 16 20
57 POGO-A 567 583 16 20
228 POGO-A 605 615 10 15
65 POGO-A 638 651 13 20
235 POGO-A 690 700 10 15
240 POGO-A 725 735 10 15
76 POGO-A 756 772 16 20
248 POGO-A 800 835 35 15
259 POGO-A 870 890 20 15
94 POGO-A 939 950 11 20
99 POGO-A 971 987 16 20
276 POGO-A 1020 1035 15 15
109 POGO-A 1062 1074 12 20
280 POGO-A 1089 1099 10 15
116 POGO-A 1123 1136 13 20
124 POGO-A 1171 1188 17 20
127 POGO-A 1221 1235 14 20
134 POGO-A 1260 1274 14 20
139 POGO-A 1318 1333 15 20
145 POGO-A 1359 1375 16 20
322 POGO-A 1415 1450 35 15
158 POGO-B 5 15 10 15
4 POGO-B 51 64 13 20
172 POGO-B 105 120 15 15
15 POGO-B 169 179 10 20
21 POGO-B 232 248 16 20
200 POGO-B 350 360 10 15
203 POGO-B 375 390 15 15
42 POGO-B 426 443 17 20
209 POGO-B 475 520 45 15
56 POGO-B 559 575 16 20
62 POGO-B 625 639 14 20
67 POGO-B 660 674 14 20
236 POGO-B 690 725 35 15
77 POGO-B 757 771 14 20
83 POGO-B 834 848 14 20
252 POGO-B 863 878 15 15
263 POGO-B 915 960 45 15
267 POGO-B 975 990 15 15
274 POGO-B 1010 1020 10 15
107 POGO-B 1047 1061 14 20
284 POGO-B 1080 1095 15 15
289 POGO-B 1110 1125 15 15
119 POGO-B 1145 1158 13 20
296 POGO-B 1173 1183 10 15
293 POGO-B 1198 1208 10 15
131 POGO-B 1247 1262 15 20
308 POGO-B 1290 1330 40 15
146 POGO-B 1362 1375 13 20
168 POGO-C 70 105 35 15
13 POGO-C 150 163 13 20
182 POGO-C 205 220 15 15
26 POGO-C 249 264 15 20
30 POGO-C 326 342 16 20
37 POGO-C 364 376 12 20
45 POGO-C 449 466 17 20
211 POGO-C 505 515 10 15
54 POGO-C 551 567 16 20
224 POGO-C 582 622 40 15
66 POGO-C 653 669 16 20
232 POGO-C 684 699 15 15
75 POGO-C 739 753 14 20
86 POGO-C 854 868 14 20
262 POGO-C 910 955 45 15
266 POGO-C 970 980 10 15
103 POGO-C 1024 1036 12 20
111 POGO-C 1071 1087 16 20
290 POGO-C 1115 1130 15 15
121 POGO-C 1161 1174 13 20
299 POGO-C 1200 1245 45 15
305 POGO-C 1265 1290 25 15
142 POGO-C 1335 1351 16 20
154 HULA-A 0 35 35 15
161 HULA-A 50 70 20 15
169 HULA-A 90 105 15 15
12 HULA-A 139 153 14 20
175 HULA-A 168 188 20 15
184 HULA-A 210 225 15 15
185 HULA-A 240 245 5 15
32 HULA-A 335 351 16 20
198 HULA-A 366 391 25 15
40 HULA-A 413 430 17 20
50 HULA-A 515 527 12 20
55 HULA-A 551 561 10 20
225 HULA-A 590 600 10 15
226 HULA-A 615 620 5 15
230 HULA-A 660 675 15 15
74 HULA-A 736 752 16 20
244 HULA-A 767 772 5 15
80 HULA-A 807 820 13 20
249 HULA-A 835 850 15 15
255 HULA-A 865 880 15 15
89 HULA-A 904 917 13 20
264 HULA-A 932 947 15 15
98 HULA-A 970 981 11 20
270 HULA-A 996 1016 20 15
277 HULA-A 1035 1040 5 15
112 HULA-A 1079 1095 16 20
288 HULA-A 1110 1120 10 15
295 HULA-A 1155 1160 5 15
298 HULA-A 1190 1205 15 15
304 HULA-A 1260 1285 25 15
312 HULA-A 1325 1335 10 15
315 HULA-A 1355 1368 13 15
160 HULA-B 30 75 45 15
24 HULA-B 238 254 16 20
33 HULA-B 335 351 16 20
199 HULA-B 366 371 5 15
212 HULA-B 510 530 20 15
221 HULA-B 560 575 15 15
73 HULA-B 736 752 16 20
239 HULA-B 767 772 5 15
81 HULA-B 807 820 13 20
254 HULA-B 855 900 45 15
97 HULA-B 970 981 11 20
102 HULA-B 1020 1036 16 20
272 HULA-B 1051 1056 5 15
113 HULA-B 1079 1095 16 20
123 HULA-B 1169 1181 12 20
128 HULA-B 1222 1234 12 20
133 HULA-B 1257 1268 11 20
309 HULA-B 1290 1305 15 15
310 HULA-B 1320 1485 165 15
156 COOK-A 0 5 5 15
5 COOK-A 54 67 13 20
174 COOK-A 145 160 15 15
188 COOK-A 240 255 15 15
194 COOK-A 300 345 45 15
44 COOK-A 441 452 11 20
216 COOK-A 525 535 10 15
219 COOK-A 550 575 25 15
231 COOK-A 660 705 45 15
242 COOK-A 735 765 30 15
250 COOK-A 825 840 15 15
88 COOK-A 874 890 16 20
261 COOK-A 905 920 15 15
100 COOK-A 975 989 14 20
282 COOK-A 1080 1095 15 15
283 COOK-A 1110 1130 20 15
294 COOK-A 1145 1160 15 15
126 COOK-A 1192 1206 14 20
138 COOK-A 1293 1308 15 20
316 COOK-A 1365 1370 5 15
151 COOK-A 1426 1441 15 20
6 COOK-B 54 68 14 20
35 COOK-B 349 364 15 20
214 COOK-B 515 525 10 15
241 COOK-B 730 925 195 15
297 COOK-B 1185 1192 7 15
136 COOK-B 1274 1290 16 20
311 COOK-B 1320 1330 10 15
152 COOK-B 1428 1443 15 20
1 INDI-A 13 28 15 20
164 INDI-A 54 64 10 15
171 INDI-A 90 110 20 15
17 INDI-A 179 195 16 20
183 INDI-A 210 250 40 15

192 INDI-A 290 300 10 15
197 INDI-A 330 335 5 15
202 INDI-A 360 375 15 15
196 INDI-A 390 410 20 15
208 INDI-A 425 515 90 15
217 INDI-A 530 545 15 15
218 INDI-A 560 570 10 15
223 INDI-A 585 600 15 15
237 INDI-A 701 711 10 15
234 INDI-A 726 736 10 15
245 INDI-A 780 785 5 15
246 INDI-A 800 810 10 15
251 INDI-A 830 850 20 15
257 INDI-A 870 890 20 15
260 INDI-A 905 910 5 15
265 INDI-A 935 955 20 15
268 INDI-A 990 995 5 15
104 INDI-A 1025 1039 14 20
279 INDI-A 1054 1059 5 15
281 INDI-A 1080 1100 20 15
292 INDI-A 1130 1140 10 15
291 INDI-A 1155 1160 5 15
300 INDI-A 1210 1220 10 15
303 INDI-A 1260 1280 20 15
141 INDI-A 1333 1349 16 20
150 INDI-A 1389 1401 12 20
320 INDI-A 1416 1436 20 15
319 INDI-A 1451 1461 10 15
159 BOSS-A 10 20 10 15
165 BOSS-A 60 360 300 15
191 BOSS-A 375 395 20 15
41 BOSS-A 421 434 13 20
48 BOSS-A 475 487 12 20
215 BOSS-A 525 530 5 15
58 BOSS-A 571 586 15 20
227 BOSS-A 601 606 5 15
222 BOSS-A 621 641 20 15
71 BOSS-A 670 686 16 20
78 BOSS-A 769 783 14 20
247 BOSS-A 800 810 10 15
258 BOSS-A 870 880 10 15
253 BOSS-A 895 900 5 15
96 BOSS-A 970 983 13 20
275 BOSS-A 1020 1040 20 15
278 BOSS-A 1055 1065 10 15
115 BOSS-A 1099 1114 15 20
118 BOSS-A 1138 1149 11 20
287 BOSS-A 1164 1174 10 15
301 BOSS-A 1215 1220 5 15
135 BOSS-A 1264 1280 16 20
306 BOSS-A 1295 1305 10 15
140 BOSS-A 1327 1340 13 20
148 BOSS-A 1369 1383 14 20
321 BOSS-A 1415 1435 20 15
157 BOSS-B 0 5 5 15
3 BOSS-B 39 55 16 20
162 BOSS-B 70 85 15 15
9 BOSS-B 117 133 16 20
14 BOSS-B 163 175 12 20
176 BOSS-B 190 200 10 15
187 BOSS-B 235 280 45 15
34 BOSS-B 340 355 15 20
220 BOSS-B 560 570 10 15
61 BOSS-B 619 635 16 20
69 BOSS-B 668 683 15 20
79 BOSS-B 771 786 15 20
243 BOSS-B 801 821 20 15
85 BOSS-B 848 863 15 20
273 BOSS-B 1005 1050 45 15
110 BOSS-B 1070 1086 16 20
286 BOSS-B 1105 1150 45 15
130 BOSS-B 1232 1247 15 20
307 BOSS-B 1290 1305 15 15
144 BOSS-B 1352 1364 12 20
317 BOSS-B 1380 1400 20 15
153 BOSS-B 1428 1442 14 20
155 LION-A 0 15 15 15
166 LION-A 60 85 25 15
173 LION-A 111 126 15 15
177 LION-A 165 170 5 15
25 LION-A 244 257 13 20
186 LION-A 272 287 15 15
29 LION-A 325 336 11 20
38 LION-A 365 381 16 20
206 LION-A 420 435 15 15
46 LION-A 466 482 16 20
213 LION-A 515 535 20 15
68 LION-A 665 678 13 20
233 LION-A 693 713 20 15
84 LION-A 845 857 12 20
256 LION-A 872 907 35 15
95 LION-A 940 955 15 20
269 LION-A 990 1010 20 15
106 LION-A 1039 1052 13 20
117 LION-A 1129 1138 9 20
132 LION-A 1252 1267 15 20
314 LION-A 1355 1390 35 15
20 LION-B 225 239 14 20
28 LION-B 271 284 13 20
195 LION-B 320 330 10 15
210 LION-B 490 510 20 15
229 LION-B 660 670 10 15
87 LION-B 867 882 15 20
91 LION-B 933 947 14 20
101 LION-B 1010 1025 15 20
108 LION-B 1052 1063 11 20
120 LION-B 1150 1166 16 20
302 LION-B 1245 1270 25 15
8 GUAM-A 108 123 15 20
178 GUAM-A 170 190 20 15
179 GUAM-A 205 230 25 15
189 GUAM-A 265 285 20 15
36 GUAM-A 355 369 14 20
207 GUAM-A 420 435 15 15
52 GUAM-A 535 550 15 20
64 GUAM-A 636 651 15 20
93 GUAM-A 935 951 16 20
105 GUAM-A 1039 1050 11 20
114 GUAM-A 1098 1112 14 20
122 GUAM-A 1168 1181 13 20
137 GUAM-A 1283 1299 16 20
313 GUAM-A 1350 1370 20 15
16 GUAM-B 173 187 14 20
190 GUAM-B 265 745 480 15
285 GUAM-B 1105 1205 100 15
147 GUAM-B 1366 1380 14 20
11 PIKE-A 138 154 16 20
23 PIKE-A 238 253 15 20
201 PIKE-A 358 415 57 15
43 PIKE-A 440 454 14 20
53 PIKE-A 539 556 17 20
60 PIKE-A 614 629 15 20
125 PIKE-A 1172 1188 16 20
7 REEF-A 81 93 12 20
18 REEF-A 193 207 14 20
181 REEF-A 222 242 20 15
49 REEF-A 480 496 16 20
59 REEF-A 582 596 14 20
63 REEF-A 632 644 12 20
72 REEF-A 728 742 14 20
82 REEF-A 831 847 16 20
92 REEF-A 933 946 13 20
129 REEF-A 1231 1248 17 20
318 REEF-A 1390 1445 55 15

Day 2

145 POGO-A 45 90 45 15
152 POGO-A 105 120 15 15
157 POGO-A 150 170 20 15
162 POGO-A 200 215 15 15
166 POGO-A 230 280 50 15
173 POGO-A 295 305 10 15
180 POGO-A 345 390 45 15
31 POGO-A 419 435 16 20
36 POGO-A 460 475 15 20
41 POGO-A 512 529 17 20
45 POGO-A 554 570 16 20
52 POGO-A 607 619 12 20
211 POGO-A 645 660 15 15
61 POGO-A 690 705 15 20
222 POGO-A 720 735 15 15
227 POGO-A 750 780 30 15
73 POGO-A 808 823 15 20
78 POGO-A 858 868 10 20
85 POGO-A 907 923 16 20
251 POGO-A 960 975 15 15

94 POGO-A 1015 1027 12 20
262 POGO-A 1065 1075 10 15
265 POGO-A 1090 1105 15 15
103 POGO-A 1132 1147 15 20
272 POGO-A 1180 1190 10 15
113 POGO-A 1232 1246 14 20
280 POGO-A 1265 1280 15 15
124 POGO-A 1331 1346 15 20
298 POGO-A 1380 1395 15 15
137 POGO-A 1431 1447 16 20
138 POGO-B 0 10 10 15
142 POGO-B 25 40 15 15
149 POGO-B 65 100 35 15
155 POGO-B 135 150 15 15
160 POGO-B 180 195 15 15
170 POGO-B 250 255 5 15
169 POGO-B 270 285 15 15
177 POGO-B 300 345 45 15
181 POGO-B 360 375 15 15
186 POGO-B 395 410 15 15
35 POGO-B 453 467 14 20
196 POGO-B 482 497 15 15
197 POGO-B 512 522 10 15
204 POGO-B 550 565 15 15
53 POGO-B 623 639 16 20
65 POGO-B 707 721 14 20
221 POGO-B 736 746 10 15
71 POGO-B 787 801 14 20
74 POGO-B 827 843 16 20
82 POGO-B 884 898 14 20
86 POGO-B 929 945 16 20
252 POGO-B 960 985 25 15
95 POGO-B 1034 1046 12 20
104 POGO-B 1133 1145 12 20
271 POGO-B 1160 1205 45 15
114 POGO-B 1233 1247 14 20
120 POGO-B 1304 1320 16 20
129 POGO-B 1362 1378 16 20
133 POGO-B 1404 1420 16 20
139 POGO-C 0 10 10 15
164 POGO-C 210 240 30 15
175 POGO-C 290 300 10 15
182 POGO-C 345 355 10 15
184 POGO-C 375 390 15 15
189 POGO-C 410 455 45 15
188 POGO-C 470 480 10 15
203 POGO-C 545 570 25 15
207 POGO-C 585 625 40 15
57 POGO-C 656 672 16 20
217 POGO-C 687 697 10 15
69 POGO-C 758 769 11 20
235 POGO-C 840 845 5 15
237 POGO-C 860 875 15 15
243 POGO-C 905 945 40 15
257 POGO-C 1005 1015 10 15
98 POGO-C 1058 1075 17 20
102 POGO-C 1131 1143 12 20
268 POGO-C 1158 1168 10 15
110 POGO-C 1205 1221 16 20
117 POGO-C 1260 1276 16 20
123 POGO-C 1323 1336 13 20
293 POGO-C 1351 1366 15 15
134 POGO-C 1420 1435 15 20
4 HULA-A 40 54 14 20
144 HULA-A 69 89 20 15
154 HULA-A 120 135 15 15
151 HULA-A 150 160 10 15
161 HULA-A 180 190 10 15
15 HULA-A 225 241 16 20
167 HULA-A 256 276 20 15
23 HULA-A 331 347 16 20
27 HULA-A 383 398 15 20
183 HULA-A 413 418 5 15
191 HULA-A 435 450 15 15
195 HULA-A 480 490 10 15
199 HULA-A 510 530 20 15
202 HULA-A 545 550 5 15
46 HULA-A 570 583 13 20
215 HULA-A 660 670 10 15
64 HULA-A 706 722 16 20
224 HULA-A 750 755 5 15
229 HULA-A 775 780 5 15
226 HULA-A 795 800 5 15
232 HULA-A 815 820 5 15
238 HULA-A 865 885 20 15
84 HULA-A 907 920 13 20
248 HULA-A 945 950 5 15
249 HULA-A 965 975 10 15
254 HULA-A 990 1010 20 15
97 HULA-A 1050 1067 17 20
100 HULA-A 1101 1116 15 20
269 HULA-A 1155 1185 30 15
275 HULA-A 1200 1215 15 15
278 HULA-A 1230 1235 5 15
288 HULA-A 1305 1315 10 15
290 HULA-A 1350 1365 15 15
299 HULA-A 1390 1405 15 15
302 HULA-A 1420 1455 35 15
143 HULA-B 60 65 5 15
158 HULA-B 160 165 5 15
21 HULA-B 307 322 15 20
29 HULA-B 408 423 15 20
37 HULA-B 482 497 15 20
209 HULA-B 595 605 10 15
223 HULA-B 730 920 190 15
245 HULA-B 935 950 15 15
91 HULA-B 1005 1022 17 20
267 HULA-B 1110 1125 15 15
270 HULA-B 1160 1170 10 15
273 HULA-B 1185 1260 75 15
285 HULA-B 1285 1295 10 15
292 HULA-B 1350 1363 13 15
140 COOK-A 16 31 15 15
16 COOK-A 239 254 15 20
19 COOK-A 282 297 15 20
24 COOK-A 334 351 17 20
187 COOK-A 405 420 15 15
194 COOK-A 470 495 25 15
220 COOK-A 720 735 15 15
228 COOK-A 770 780 10 15
230 COOK-A 795 830 35 15
236 COOK-A 845 860 15 15
240 COOK-A 880 890 10 15
88 COOK-A 946 962 16 20
258 COOK-A 1020 1210 190 15
115 COOK-A 1244 1260 16 20
286 COOK-A 1290 1330 40 15
130 COOK-A 1364 1375 11 20
14 COOK-B 209 226 17 20
185 COOK-B 390 410 20 15
193 COOK-B 455 490 35 15
219 COOK-B 690 725 35 15
246 COOK-B 915 960 45 15
253 COOK-B 980 1125 145 15
277 COOK-B 1210 1290 80 15
279 COOK-B 1305 1310 5 15
126 COOK-B 1346 1359 13 20
141 INDI-A 5 10 5 15
147 INDI-A 45 119 74 15
156 INDI-A 140 145 5 15
159 INDI-A 165 190 25 15
163 INDI-A 205 225 20 15
168 INDI-A 240 245 5 15
171 INDI-A 265 285 20 15
178 INDI-A 320 330 10 15
176 INDI-A 345 355 10 15
179 INDI-A 370 390 20 15
190 INDI-A 430 435 5 15
192 INDI-A 450 470 20 15
43 INDI-A 550 566 16 20
210 INDI-A 600 605 5 15
213 INDI-A 655 665 10 15
60 INDI-A 687 700 13 20
218 INDI-A 715 735 20 15
231 INDI-A 800 810 10 15
234 INDI-A 835 840 5 15
239 INDI-A 870 890 20 15
241 INDI-A 905 915 10 15
247 INDI-A 935 955 20 15
93 INDI-A 1007 1019 12 20
261 INDI-A 1050 1055 5 15
259 INDI-A 1070 1075 5 15
266 INDI-A 1105 1200 95 15
119 INDI-A 1302 1319 17 20
283 INDI-A 1334 1344 10 15
295 INDI-A 1375 1380 5 15
132 INDI-A 1402 1413 11 20
301 INDI-A 1428 1448 20 15
5 BOSS-A 46 61 15 20
6 BOSS-A 87 100 13 20
8 BOSS-A 141 158 17 20
12 BOSS-A 189 199 10 20

```
165 BOSS-A 225 230 5 15
174 BOSS-A 270 290 20 15
22 BOSS-A 310 324 14 20
30 BOSS-A 410 426 16 20
34 BOSS-A 452 466 14 20
198 BOSS-A 510 520 10 15
205 BOSS-A 560 570 10 15
49 BOSS-A 604 621 17 20
55 BOSS-A 641 656 15 20
206 BOSS-A 671 691 20 15
214 BOSS-A 706 716 10 15
68 BOSS-A 742 758 16 20
225 BOSS-A 773 793 20 15
242 BOSS-A 900 910 10 15
87 BOSS-A 940 951 11 20
250 BOSS-A 966 981 15 15
96 BOSS-A 1040 1056 16 20
107 BOSS-A 1165 1177 12 20
274 BOSS-A 1195 1200 5 15
282 BOSS-A 1275 1290 15 15
297 BOSS-A 1380 1400 20 15
135 BOSS-A 1422 1438 16 20
148 BOSS-B 60 360 300 15
38 BOSS-B 505 518 13 20
48 BOSS-B 602 616 14 20
208 BOSS-B 631 636 5 15
216 BOSS-B 675 680 5 15
244 BOSS-B 915 930 15 15
260 BOSS-B 1040 1050 10 15
263 BOSS-B 1070 1090 20 15
281 BOSS-B 1270 1295 25 15
289 BOSS-B 1320 1485 165 15
2 LION-A 11 24 13 20
153 LION-A 110 120 10 15
150 LION-A 135 155 20 15
13 LION-A 209 222 13 20
18 LION-A 257 271 14 20
25 LION-A 336 351 15 20
28 LION-A 404 414 10 20
32 LION-A 437 453 16 20
42 LION-A 540 551 11 20
54 LION-A 635 646 11 20
62 LION-A 693 706 13 20
67 LION-A 737 749 12 20
76 LION-A 837 852 15 20
81 LION-A 875 888 13 20
90 LION-A 971 985 14 20
255 LION-A 1000 1020 20 15
264 LION-A 1080 1100 20 15
101 LION-A 1122 1137 15 20
276 LION-A 1209 1219 10 15
287 LION-A 1295 1305 10 15
131 LION-A 1369 1384 15 20
136 LION-A 1422 1433 11 20
300 LION-A 1448 1458 10 15
146 LION-B 45 480 435 15
201 LION-B 525 550 25 15
212 LION-B 655 670 15 15
66 LION-B 735 751 16 20
80 LION-B 868 880 12 20
89 LION-B 965 980 15 20
99 LION-B 1063 1075 12 20
111 LION-B 1223 1239 16 20
284 LION-B 1280 1290 10 15
121 LION-B 1314 1330 16 20
294 LION-B 1355 1445 90 15
9 GUAM-A 143 156 13 20
17 GUAM-A 243 259 16 20
26 GUAM-A 373 387 14 20
39 GUAM-A 507 521 14 20
51 GUAM-A 607 623 16 20
59 GUAM-A 669 684 15 20
70 GUAM-A 766 781 15 20
233 GUAM-A 830 850 20 15
83 GUAM-A 905 921 16 20
92 GUAM-A 1007 1021 14 20
108 GUAM-A 1200 1209 9 20
116 GUAM-A 1254 1270 16 20
127 GUAM-A 1357 1370 13 20
291 GUAM-A 1385 1405 20 15
172 GUAM-B 265 745 480 15
256 GUAM-B 1000 1660 660 15
7 PIKE-A 110 126 16 20
11 PIKE-A 167 182 15 20
40 PIKE-A 509 525 16 20
44 PIKE-A 550 562 12 20
50 PIKE-A 605 617 12 20
56 PIKE-A 646 660 14 20
63 PIKE-A 701 716 15 20
77 PIKE-A 844 860 16 20
105 PIKE-A 1142 1157 15 20
112 PIKE-A 1225 1240 15 20
118 PIKE-A 1266 1276 10 20
128 PIKE-A 1360 1375 15 20
296 PIKE-A 1390 1410 20 15
10 REEF-A 151 167 16 20
33 REEF-A 451 465 14 20
200 REEF-A 515 535 20 15
47 REEF-A 591 602 11 20
58 REEF-A 661 676 15 20
72 REEF-A 803 818 15 20
79 REEF-A 859 872 13 20
109 REEF-A 1201 1217 16 20
122 REEF-A 1322 1334 12 20
```

Day 3

```
166 POGO-A 1 1 0 0
152 POGO-A 30 60 30 15
8 POGO-A 92 108 16 20
164 POGO-A 135 150 15 15
19 POGO-A 197 213 16 20
180 POGO-A 240 255 15 15
25 POGO-A 288 304 16 20
30 POGO-A 330 342 12 20
195 POGO-A 360 375 15 15
37 POGO-A 397 414 17 20
42 POGO-A 440 454 14 20
45 POGO-A 490 506 16 20
215 POGO-A 540 555 15 15
52 POGO-A 575 586 11 20
64 POGO-A 676 689 13 20
72 POGO-A 721 735 14 20
234 POGO-A 750 770 20 15
79 POGO-A 804 818 14 20
238 POGO-A 833 838 5 15
244 POGO-A 855 870 15 15
89 POGO-A 899 915 16 20
251 POGO-A 930 945 15 15
257 POGO-A 970 1030 60 15
266 POGO-A 1045 1060 15 15
273 POGO-A 1080 1090 10 15
111 POGO-A 1116 1128 12 20
277 POGO-A 1143 1153 10 15
117 POGO-A 1174 1190 16 20
283 POGO-A 1205 1215 10 15
289 POGO-A 1235 1250 15 15
127 POGO-A 1273 1289 16 20
133 POGO-A 1320 1335 15 20
310 POGO-A 1405 1440 35 15
2 POGO-B 7 21 14 20
5 POGO-B 64 80 16 20
159 POGO-B 95 110 15 15
12 POGO-B 131 142 11 20
14 POGO-B 165 180 15 20
170 POGO-B 195 205 10 15
26 POGO-B 297 313 16 20
32 POGO-B 337 350 13 20
36 POGO-B 389 405 16 20
41 POGO-B 428 441 13 20
201 POGO-B 456 466 10 15
46 POGO-B 498 514 16 20
51 POGO-B 541 557 16 20
56 POGO-B 592 608 16 20
62 POGO-B 643 659 16 20
68 POGO-B 695 710 15 20
232 POGO-B 725 735 10 15
75 POGO-B 776 791 15 20
82 POGO-B 817 832 15 20
246 POGO-B 860 880 20 15
250 POGO-B 900 910 10 15
254 POGO-B 930 945 15 15
264 POGO-B 1000 1010 10 15
104 POGO-B 1045 1062 17 20
109 POGO-B 1105 1117 12 20
```

113	POGO-B	1146	1162	16	20
121	POGO-B	1218	1232	14	20
290	POGO-B	1247	1257	10	15
128	POGO-B	1289	1303	14	20
139	POGO-B	1350	1366	16	20
151	POGO-C	30	45	15	15
7	POGO-C	90	103	13	20
163	POGO-C	120	135	15	15
17	POGO-C	189	203	14	20
186	POGO-C	285	295	10	15
196	POGO-C	360	375	15	15
199	POGO-C	390	405	15	15
205	POGO-C	420	435	15	15
47	POGO-C	501	517	16	20
216	POGO-C	540	555	15	15
57	POGO-C	599	615	16	20
69	POGO-C	710	721	11	20
78	POGO-C	797	813	16	20
248	POGO-C	900	945	45	15
258	POGO-C	975	990	15	15
269	POGO-C	1060	1070	10	15
108	POGO-C	1101	1117	16	20
279	POGO-C	1155	1165	10	15
285	POGO-C	1225	1270	45	15
131	POGO-C	1301	1314	13	20
138	POGO-C	1347	1360	13	20
148	HULA-A	30	45	15	15
158	HULA-A	60	95	35	15
10	HULA-A	122	136	14	20
168	HULA-A	175	185	10	15
21	HULA-A	212	229	17	20
181	HULA-A	260	265	5	15
178	HULA-A	280	300	20	15
185	HULA-A	315	320	5	15
192	HULA-A	340	350	10	15
194	HULA-A	365	375	10	15
38	HULA-A	398	408	10	20
43	HULA-A	451	467	16	20
210	HULA-A	495	570	75	15
217	HULA-A	585	590	5	15
65	HULA-A	676	691	15	20
231	HULA-A	710	720	10	15
237	HULA-A	780	785	5	15
86	HULA-A	869	884	15	20
252	HULA-A	915	920	5	15
249	HULA-A	935	940	5	15
256	HULA-A	955	985	30	15
101	HULA-A	1022	1038	16	20
271	HULA-A	1071	1086	15	15
112	HULA-A	1124	1137	13	20
280	HULA-A	1170	1200	30	15
287	HULA-A	1230	1240	10	15
292	HULA-A	1260	1275	15	15
288	HULA-A	1290	1295	5	15
302	HULA-A	1325	1330	5	15
303	HULA-A	1345	1359	14	15
147	HULA-B	0	10	10	15
153	HULA-B	30	50	20	15
165	HULA-B	145	165	20	15
174	HULA-B	195	210	15	15
179	HULA-B	240	255	15	15
188	HULA-B	300	345	45	15
34	HULA-B	379	395	16	20
212	HULA-B	510	530	20	15
220	HULA-B	565	605	40	15
230	HULA-B	705	750	45	15
76	HULA-B	778	792	14	20
96	HULA-B	991	1007	16	20
102	HULA-B	1034	1048	14	20
275	HULA-B	1105	1120	15	15
281	HULA-B	1175	1184	9	15
124	HULA-B	1245	1258	13	20
298	HULA-B	1285	1325	40	15
4	COOK-A	20	34	14	20
167	COOK-A	175	220	45	15
29	COOK-A	315	330	15	20
197	COOK-A	375	390	15	15
204	COOK-A	420	435	15	15
211	COOK-A	495	505	10	15
208	COOK-A	520	525	5	15
214	COOK-A	540	565	25	15
222	COOK-A	580	590	10	15
229	COOK-A	690	705	15	15
239	COOK-A	790	825	35	15
85	COOK-A	858	874	16	20
92	COOK-A	917	933	16	20
261	COOK-A	990	1010	20	15
268	COOK-A	1051	1061	10	15
106	COOK-A	1089	1103	14	20
114	COOK-A	1152	1166	14	20
282	COOK-A	1190	1200	10	15
295	COOK-A	1267	1307	40	15
145	COOK-A	1395	1408	13	20
16	COOK-B	181	197	16	20
182	COOK-B	265	740	475	15
84	COOK-B	833	847	14	20
260	COOK-B	980	1170	190	15
294	COOK-B	1260	1285	25	15
132	COOK-B	1315	1329	14	20
142	COOK-B	1391	1404	13	20
154	INDI-A	30	35	5	15
156	INDI-A	52	62	10	15
160	INDI-A	80	90	10	15
169	INDI-A	175	180	5	15
176	INDI-A	200	220	20	15
183	INDI-A	265	285	20	15
189	INDI-A	320	330	10	15
190	INDI-A	345	365	20	15
202	INDI-A	420	425	5	15
206	INDI-A	450	470	20	15
49	INDI-A	520	535	15	20
219	INDI-A	560	570	10	15
223	INDI-A	600	610	10	15
224	INDI-A	630	640	10	15
225	INDI-A	655	665	10	15
233	INDI-A	720	725	5	15
240	INDI-A	800	815	15	15
242	INDI-A	830	850	20	15
247	INDI-A	870	890	20	15
255	INDI-A	935	955	20	15
95	INDI-A	977	991	14	20
259	INDI-A	1006	1011	5	15
267	INDI-A	1040	1050	10	15
270	INDI-A	1070	1085	15	15
276	INDI-A	1110	1135	25	15
284	INDI-A	1208	1218	10	15
125	INDI-A	1272	1288	16	20
301	INDI-A	1320	1330	10	15
306	INDI-A	1370	1375	5	15
309	INDI-A	1395	1405	10	15
311	INDI-A	1420	1440	20	15
149	BOSS-A	15	60	45	15
9	BOSS-A	100	115	15	20
13	BOSS-A	164	180	16	20
175	BOSS-A	195	210	15	15
24	BOSS-A	281	293	12	20
184	BOSS-A	308	328	20	15
35	BOSS-A	379	396	17	20
44	BOSS-A	483	497	14	20
50	BOSS-A	535	549	14	20
55	BOSS-A	582	594	12	20
60	BOSS-A	634	649	15	20
67	BOSS-A	679	693	14	20
70	BOSS-A	713	730	17	20
227	BOSS-A	745	755	10	15
236	BOSS-A	770	790	20	15
243	BOSS-A	850	885	35	15
93	BOSS-A	959	971	12	20
97	BOSS-A	995	995	0	20
265	BOSS-A	1020	1040	20	15
278	BOSS-A	1125	1130	5	15
119	BOSS-A	1196	1210	14	20
286	BOSS-A	1230	1240	10	15
293	BOSS-A	1260	1265	5	15
129	BOSS-A	1293	1307	14	20
144	BOSS-A	1393	1409	16	20
307	BOSS-A	1424	1444	20	15
162	BOSS-B	115	190	75	15
171	BOSS-B	205	215	10	15
177	BOSS-B	230	280	50	15
193	BOSS-B	345	390	45	15
209	BOSS-B	490	505	15	15
218	BOSS-B	555	570	15	15
58	BOSS-B	613	627	14	20
221	BOSS-B	642	662	20	15
226	BOSS-B	685	720	35	15
245	BOSS-B	860	865	5	15
88	BOSS-B	885	900	15	20
253	BOSS-B	930	940	10	15
100	BOSS-B	1010	1025	15	20
123	BOSS-B	1239	1255	16	20
297	BOSS-B	1280	1290	10	15

141 BOSS-B 1369 1379 10 20
308 BOSS-B 1394 1414 20 15
161 LION-A 85 105 20 15
18 LION-A 191 205 14 20
23 LION-A 239 253 14 20
27 LION-A 308 322 14 20
191 LION-A 337 347 10 15
198 LION-A 375 380 5 15
203 LION-A 420 480 60 15
48 LION-A 511 524 13 20
228 LION-A 690 710 20 15
80 LION-A 806 822 16 20
91 LION-A 904 919 15 20
98 LION-A 997 1011 14 20
262 LION-A 1026 1046 20 15
107 LION-A 1094 1108 14 20
118 LION-A 1194 1210 16 20
130 LION-A 1298 1311 13 20
143 LION-A 1393 1402 9 20
150 LION-B 20 25 5 15
157 LION-B 60 180 120 15
173 LION-B 195 210 15 15
31 LION-B 336 350 14 20
200 LION-B 390 420 30 15
207 LION-B 480 525 45 15
241 LION-B 810 845 35 15
90 LION-B 900 913 13 20
99 LION-B 1002 1016 14 20
272 LION-B 1080 1100 20 15
300 LION-B 1300 1310 10 15
305 LION-B 1365 1405 40 15
15 GUAM-A 171 183 12 20
20 GUAM-A 212 229 17 20
28 GUAM-A 308 319 11 20
39 GUAM-A 403 417 14 20
53 GUAM-A 578 594 16 20
63 GUAM-A 645 661 16 20
77 GUAM-A 797 810 13 20
94 GUAM-A 976 991 15 20
105 GUAM-A 1064 1078 14 20
274 GUAM-A 1105 1200 95 15
122 GUAM-A 1226 1241 15 20
291 GUAM-A 1260 1280 20 15
299 GUAM-A 1295 1305 10 15
135 GUAM-A 1327 1342 15 20
304 GUAM-A 1357 1377 20 15
235 GUAM-B 725 915 190 15
103 GUAM-B 1036 1046 10 20
296 GUAM-B 1275 1290 15 15
136 GUAM-B 1332 1346 14 20
3 PIKE-A 19 30 11 20
6 PIKE-A 82 97 15 20
172 PIKE-A 180 360 180 15
54 PIKE-A 581 595 14 20
61 PIKE-A 635 648 13 20
66 PIKE-A 678 692 14 20
73 PIKE-A 732 746 14 20
81 PIKE-A 815 832 17 20
263 PIKE-A 990 1035 45 15
110 PIKE-A 1112 1126 14 20
115 PIKE-A 1154 1169 15 20
120 PIKE-A 1212 1228 16 20
134 PIKE-A 1320 1365 45 20
11 REEF-A 123 139 16 20
22 REEF-A 226 237 11 20
187 REEF-A 300 320 20 15
33 REEF-A 357 372 15 20
40 REEF-A 422 433 11 20
213 REEF-A 515 535 20 15
59 REEF-A 621 635 14 20
71 REEF-A 719 731 12 20
74 REEF-A 776 788 12 20
87 REEF-A 875 891 16 20
116 REEF-A 1171 1186 15 20
126 REEF-A 1272 1287 15 20
137 REEF-A 1335 1347 12 20
146 REEF-A 1431 1445 14 20

147 POGO-A 15 35 20 15
6 POGO-A 61 73 12 20
160 POGO-A 90 105 15 15
12 POGO-A 159 173 14 20
169 POGO-A 190 205 15 15
20 POGO-A 232 246 14 20
24 POGO-A 268 283 15 20
183 POGO-A 300 345 45 15
35 POGO-A 383 399 16 20
40 POGO-A 427 441 14 20
45 POGO-A 483 500 17 20
205 POGO-A 515 530 15 15
51 POGO-A 556 571 15 20
215 POGO-A 586 596 10 15
57 POGO-A 623 636 13 20
62 POGO-A 664 680 16 20
231 POGO-A 695 700 5 15
70 POGO-A 731 744 13 20
73 POGO-A 766 782 16 20
79 POGO-A 823 834 11 20
84 POGO-A 866 881 15 20
256 POGO-A 910 925 15 15
263 POGO-A 950 975 25 15
269 POGO-A 995 1005 10 15
100 POGO-A 1032 1049 17 20
106 POGO-A 1077 1089 12 20
111 POGO-A 1133 1150 17 20
288 POGO-A 1165 1175 10 15
116 POGO-A 1203 1217 14 20
122 POGO-A 1274 1289 15 20
304 POGO-A 1304 1314 10 15
135 POGO-A 1374 1390 16 20
316 POGO-A 1405 1440 35 15
143 POGO-B 0 15 15 15
154 POGO-B 60 105 45 15
9 POGO-B 135 149 14 20
15 POGO-B 185 202 17 20
26 POGO-B 283 299 16 20
28 POGO-B 324 336 12 20
33 POGO-B 361 373 12 20
194 POGO-B 395 405 10 15
195 POGO-B 420 435 15 15
42 POGO-B 459 476 17 20
199 POGO-B 491 506 15 15
49 POGO-B 528 544 16 20
53 POGO-B 574 591 17 20
221 POGO-B 606 621 15 15
63 POGO-B 664 680 16 20
71 POGO-B 745 759 14 20
77 POGO-B 789 804 15 20
81 POGO-B 845 860 15 20
87 POGO-B 894 906 12 20
93 POGO-B 945 961 16 20
270 POGO-B 1000 1020 20 15
274 POGO-B 1035 1050 15 15
278 POGO-B 1065 1075 10 15
285 POGO-B 1120 1135 15 15
114 POGO-B 1172 1187 15 20
118 POGO-B 1234 1251 17 20
124 POGO-B 1279 1291 12 20
130 POGO-B 1341 1357 16 20
138 POGO-B 1406 1423 17 20
146 POGO-C 15 25 10 15
153 POGO-C 55 90 35 15
163 POGO-C 105 140 35 15
13 POGO-C 163 173 10 20
166 POGO-C 188 203 15 15
22 POGO-C 258 274 16 20
179 POGO-C 289 299 10 15
184 POGO-C 315 320 5 15
32 POGO-C 358 375 17 20
190 POGO-C 390 405 15 15
39 POGO-C 426 441 15 20
201 POGO-C 480 525 45 15
52 POGO-C 561 578 17 20
60 POGO-C 645 657 12 20
227 POGO-C 680 690 10 15
234 POGO-C 720 730 10 15
72 POGO-C 751 765 14 20
238 POGO-C 780 815 35 15
240 POGO-C 830 845 15 15
249 POGO-C 860 880 20 15
254 POGO-C 910 920 10 15
96 POGO-C 977 990 13 20
102 POGO-C 1041 1055 14 20

Day 4

284	POGO-C	1110	1155	45	15
287	POGO-C	1170	1185	15	15
293	POGO-C	1200	1215	15	15
120	POGO-C	1247	1263	16	20
127	POGO-C	1305	1320	15	20
134	POGO-C	1369	1382	13	20
139	POGO-C	1407	1420	13	20
149	HULA-A	25	40	15	15
155	HULA-A	60	70	10	15
150	HULA-A	85	105	20	15
10	HULA-A	142	157	15	20
17	HULA-A	199	216	17	20
167	HULA-A	231	236	5	15
174	HULA-A	251	271	20	15
181	HULA-A	290	295	5	15
188	HULA-A	345	390	45	15
38	HULA-A	419	430	11	20
202	HULA-A	490	500	10	15
48	HULA-A	522	533	11	20
218	HULA-A	580	600	20	15
211	HULA-A	615	635	20	15
225	HULA-A	660	670	10	15
232	HULA-A	705	720	15	15
233	HULA-A	735	750	15	15
78	HULA-A	804	818	14	20
247	HULA-A	850	925	75	15
258	HULA-A	940	945	5	15
95	HULA-A	976	993	17	20
99	HULA-A	1032	1047	15	20
107	HULA-A	1079	1091	12	20
286	HULA-A	1125	1130	5	15
291	HULA-A	1185	1190	5	15
296	HULA-A	1230	1275	45	15
302	HULA-A	1290	1330	40	15
136	HULA-A	1377	1390	13	20
309	HULA-A	1405	1415	10	15
318	HULA-A	1430	1445	15	15
158	HULA-B	70	80	10	15
164	HULA-B	140	160	20	15
172	HULA-B	225	230	5	15
170	HULA-B	245	255	10	15
180	HULA-B	285	305	20	15
187	HULA-B	340	350	10	15
191	HULA-B	385	395	10	15
37	HULA-B	419	436	17	20
47	HULA-B	515	526	11	20
213	HULA-B	565	605	40	15
228	HULA-B	680	695	15	15
235	HULA-B	720	735	15	15
241	HULA-B	820	825	5	15
83	HULA-B	866	880	14	20
90	HULA-B	902	915	13	20
261	HULA-B	950	960	10	15
98	HULA-B	994	1009	15	20
266	HULA-B	1024	1044	20	15
109	HULA-B	1095	1110	15	20
289	HULA-B	1155	1180	25	15
294	HULA-B	1200	1245	45	15
123	HULA-B	1275	1290	15	20
300	HULA-B	1305	1320	15	15
307	HULA-B	1345	1360	15	15
4	COOK-A	52	65	13	20
165	COOK-A	160	165	5	15
19	COOK-A	227	241	14	20
175	COOK-A	256	261	5	15
29	COOK-A	349	363	14	20
189	COOK-A	378	393	15	15
209	COOK-A	540	550	10	15
219	COOK-A	585	600	15	15
229	COOK-A	680	715	35	15
244	COOK-A	840	850	10	15
86	COOK-A	889	905	16	20
257	COOK-A	925	955	30	15
264	COOK-A	970	975	5	15
268	COOK-A	990	1035	45	15
103	COOK-A	1058	1073	15	20
283	COOK-A	1105	1120	15	15
297	COOK-A	1245	1260	15	15
125	COOK-A	1284	1300	16	20
141	COOK-A	1426	1440	14	20
3	COOK-B	48	61	13	20
173	COOK-B	225	275	50	15
193	COOK-B	390	415	25	15
207	COOK-B	530	545	15	15
217	COOK-B	570	580	10	15
226	COOK-B	660	675	15	15
248	COOK-B	850	865	15	15
253	COOK-B	900	945	45	15
262	COOK-B	960	970	10	15
267	COOK-B	990	1655	665	15
144	INDI-A	30	35	5	15
152	INDI-A	53	63	10	15
161	INDI-A	90	100	10	15
159	INDI-A	115	135	20	15
168	INDI-A	180	185	5	15
171	INDI-A	220	265	45	15
182	INDI-A	300	305	5	15
185	INDI-A	320	330	10	15
186	INDI-A	345	365	20	15
192	INDI-A	390	415	25	15
198	INDI-A	450	470	20	15
203	INDI-A	500	505	5	15
206	INDI-A	520	540	20	15
212	INDI-A	560	570	10	15
210	INDI-A	585	595	10	15
224	INDI-A	660	665	5	15
223	INDI-A	680	700	20	15
68	INDI-A	720	735	15	20
239	INDI-A	800	810	10	15
242	INDI-A	830	850	20	15
251	INDI-A	870	890	20	15
250	INDI-A	905	925	20	15
94	INDI-A	948	964	16	20
259	INDI-A	979	989	10	15
273	INDI-A	1020	1025	5	15
272	INDI-A	1040	1050	10	15
276	INDI-A	1065	1080	15	15
282	INDI-A	1105	1200	95	15
292	INDI-A	1215	1235	20	15
298	INDI-A	1255	1280	25	15
301	INDI-A	1295	1305	10	15
131	INDI-A	1344	1357	13	20
137	INDI-A	1387	1399	12	20
315	INDI-A	1414	1424	10	15
311	INDI-A	1439	1444	5	15
5	BOSS-A	54	70	16	20
157	BOSS-A	85	90	5	15
8	BOSS-A	125	140	15	20
21	BOSS-A	252	262	10	20
177	BOSS-A	277	297	20	15
30	BOSS-A	349	365	16	20
36	BOSS-A	419	431	12	20
43	BOSS-A	469	481	12	20
204	BOSS-A	510	540	30	15
55	BOSS-A	584	597	13	20
214	BOSS-A	612	632	20	15
65	BOSS-A	667	682	15	20
236	BOSS-A	750	770	20	15
245	BOSS-A	840	850	10	15
252	BOSS-A	885	890	5	15
97	BOSS-A	979	994	15	20
271	BOSS-A	1020	1040	20	15
275	BOSS-A	1055	1065	10	15
279	BOSS-A	1080	1090	10	15
281	BOSS-A	1105	1120	15	15
117	BOSS-A	1226	1241	15	20
303	BOSS-A	1285	1295	10	15
128	BOSS-A	1326	1342	16	20
314	BOSS-A	1380	1385	5	15
142	BOSS-A	1434	1448	14	20
2	BOSS-B	24	40	16	20
156	BOSS-B	60	360	300	15
196	BOSS-B	425	440	15	15
200	BOSS-B	480	485	5	15
54	BOSS-B	576	592	16	20
220	BOSS-B	607	617	10	15
59	BOSS-B	639	656	17	20
67	BOSS-B	685	701	16	20
237	BOSS-B	760	775	15	15
243	BOSS-B	840	1320	480	15
133	BOSS-B	1366	1380	14	20
312	BOSS-B	1395	1415	20	15
145	LION-A	10	15	5	15
162	LION-A	90	115	25	15
18	LION-A	223	237	14	20
25	LION-A	269	284	15	20
34	LION-A	380	396	16	20
197	LION-A	450	455	5	15
44	LION-A	482	496	14	20
208	LION-A	540	565	25	15
222	LION-A	605	625	20	15
66	LION-A	675	689	14	20

230 LION-A 704 724 20 15
74 LION-A 776 792 16 20
80 LION-A 839 851 12 20
88 LION-A 898 913 15 20
260 LION-A 935 955 20 15
101 LION-A 1033 1046 13 20
104 LION-A 1066 1079 13 20
110 LION-A 1127 1136 9 20
290 LION-A 1170 1190 20 15
295 LION-A 1208 1218 10 15
121 LION-A 1268 1282 14 20
310 LION-A 1360 1375 15 15
317 LION-A 1410 1430 20 15
151 LION-B 45 480 435 15
216 LION-B 570 595 25 15
85 LION-B 878 890 12 20
91 LION-B 931 945 14 20
265 LION-B 990 1035 45 15
280 LION-B 1080 1100 20 15
113 LION-B 1165 1181 16 20
306 LION-B 1340 1360 20 15
313 LION-B 1380 1435 55 15
14 GUAM-A 182 198 16 20
27 GUAM-A 285 297 12 20
31 GUAM-A 352 367 15 20
61 GUAM-A 653 664 11 20
69 GUAM-A 726 741 15 20
246 GUAM-A 840 885 45 15
92 GUAM-A 945 961 16 20
277 GUAM-A 1055 1170 115 15
126 GUAM-A 1298 1314 16 20
308 GUAM-A 1350 1370 20 15
23 GUAM-B 260 315 55 20
89 GUAM-B 900 940 40 20
299 GUAM-B 1260 1280 20 15
305 GUAM-B 1320 1485 165 15
11 PIKE-A 153 170 17 20
176 PIKE-A 245 255 10 15
41 PIKE-A 449 464 15 20
50 PIKE-A 550 565 15 20
56 PIKE-A 612 626 14 20
64 PIKE-A 665 679 14 20
76 PIKE-A 787 803 16 20
108 PIKE-A 1082 1094 12 20
115 PIKE-A 1182 1198 16 20
129 PIKE-A 1327 1341 14 20
140 PIKE-A 1422 1435 13 20
148 REEF-A 20 40 20 15
7 REEF-A 94 110 16 20
16 REEF-A 196 210 14 20
178 REEF-A 270 290 20 15
46 REEF-A 490 506 16 20
58 REEF-A 626 638 12 20
82 REEF-A 846 862 16 20
255 REEF-A 910 930 20 15
112 REEF-A 1142 1155 13 20
119 REEF-A 1242 1257 15 20
132 REEF-A 1364 1378 14 20

Day 5

3 POGO-A 31 43 12 20
144 POGO-A 58 73 15 15
8 POGO-A 97 109 12 20
12 POGO-A 136 153 17 20
166 POGO-A 185 200 15 15
21 POGO-A 228 243 15 20
24 POGO-A 268 284 16 20
32 POGO-A 360 375 15 20
37 POGO-A 414 427 13 20
45 POGO-A 469 485 16 20
49 POGO-A 516 531 15 20
213 POGO-A 570 580 10 15
59 POGO-A 613 625 12 20
221 POGO-A 640 650 10 15
229 POGO-A 675 690 15 15
232 POGO-A 705 715 10 15
235 POGO-A 730 745 15 15
76 POGO-A 775 789 14 20
79 POGO-A 814 829 15 20
86 POGO-A 879 891 12 20
87 POGO-A 911 922 11 20
92 POGO-A 947 960 13 20
253 POGO-A 975 990 15 15
261 POGO-A 1005 1020 15 15
103 POGO-A 1048 1061 13 20
270 POGO-A 1100 1115 15 15
273 POGO-A 1130 1140 10 15
276 POGO-A 1165 1175 10 15
120 POGO-A 1211 1227 16 20
285 POGO-A 1255 1280 25 15
129 POGO-A 1310 1326 16 20
295 POGO-A 1350 1365 15 15
139 POGO-A 1410 1426 16 20
143 POGO-B 0 10 10 15
4 POGO-B 35 51 16 20
153 POGO-B 66 81 15 15
156 POGO-B 96 101 5 15
11 POGO-B 129 143 14 20
16 POGO-B 167 181 14 20
20 POGO-B 204 217 13 20
23 POGO-B 263 278 15 20
176 POGO-B 293 303 10 15
30 POGO-B 328 344 16 20
36 POGO-B 391 404 13 20
40 POGO-B 429 445 16 20
46 POGO-B 489 503 14 20
198 POGO-B 518 523 5 15
52 POGO-B 545 562 17 20
56 POGO-B 587 602 15 20
62 POGO-B 633 649 16 20
68 POGO-B 684 699 15 20
77 POGO-B 781 795 14 20
82 POGO-B 838 854 16 20
240 POGO-B 869 899 30 15
251 POGO-B 914 929 15 15
95 POGO-B 955 966 11 20
260 POGO-B 990 1000 10 15
100 POGO-B 1020 1036 16 20
107 POGO-B 1071 1086 15 20
109 POGO-B 1120 1137 17 20
275 POGO-B 1152 1162 10 15
115 POGO-B 1189 1202 13 20
123 POGO-B 1246 1260 14 20
127 POGO-B 1290 1305 15 20
292 POGO-B 1335 1350 15 15
137 POGO-B 1392 1408 16 20
304 POGO-B 1423 1433 10 15
6 POGO-C 67 84 17 20
160 POGO-C 105 120 15 15
17 POGO-C 168 184 16 20
169 POGO-C 220 275 55 15
182 POGO-C 330 345 15 15
190 POGO-C 375 395 20 15
195 POGO-C 420 435 15 15
192 POGO-C 450 460 10 15
199 POGO-C 480 495 15 15
51 POGO-C 531 547 16 20
220 POGO-C 580 595 15 15
63 POGO-C 642 658 16 20
228 POGO-C 673 683 10 15
70 POGO-C 714 728 14 20
73 POGO-C 748 764 16 20
243 POGO-C 845 860 15 15
88 POGO-C 914 929 15 20
93 POGO-C 949 962 13 20
254 POGO-C 977 1007 30 15
267 POGO-C 1050 1060 10 15
113 POGO-C 1152 1166 14 20
125 POGO-C 1256 1268 12 20
289 POGO-C 1283 1323 40 15
133 POGO-C 1346 1361 15 20
301 POGO-C 1395 1430 35 15
149 HULA-A 30 50 20 15
146 HULA-A 65 70 5 15
164 HULA-A 135 155 20 15
18 HULA-A 187 203 16 20
170 HULA-A 240 260 20 15
178 HULA-A 280 295 15 15
171 HULA-A 310 325 15 15
185 HULA-A 340 350 10 15
35 HULA-A 389 404 15 20
39 HULA-A 426 437 11 20
202 HULA-A 485 495 10 15
204 HULA-A 510 530 20 15
215 HULA-A 570 580 10 15

216 HULA-A 595 605 10 15
224 HULA-A 630 635 5 15
225 HULA-A 650 665 15 15
71 HULA-A 716 732 16 20
237 HULA-A 765 800 35 15
81 HULA-A 835 850 15 20
249 HULA-A 895 940 45 15
97 HULA-A 962 978 16 20
99 HULA-A 1000 1014 14 20
256 HULA-A 1029 1044 15 15
106 HULA-A 1064 1077 13 20
272 HULA-A 1110 1115 5 15
269 HULA-A 1130 1135 5 15
274 HULA-A 1150 1175 25 15
118 HULA-A 1205 1218 13 20
281 HULA-A 1233 1238 5 15
284 HULA-A 1255 1265 10 15
128 HULA-A 1305 1320 15 20
135 HULA-A 1358 1374 16 20
297 HULA-A 1389 1399 10 15
151 HULA-B 45 80 35 15
161 HULA-B 125 140 15 15
165 HULA-B 180 195 15 15
25 HULA-B 289 302 13 20
29 HULA-B 322 338 16 20
34 HULA-B 375 390 15 20
187 HULA-B 405 420 15 15
47 HULA-B 489 503 14 20
207 HULA-B 535 560 25 15
210 HULA-B 575 615 40 15
214 HULA-B 630 640 10 15
227 HULA-B 665 700 35 15
233 HULA-B 715 720 5 15
238 HULA-B 775 810 35 15
244 HULA-B 850 870 20 15
248 HULA-B 890 900 10 15
258 HULA-B 990 1010 20 15
280 HULA-B 1190 1200 10 15
290 HULA-B 1290 1295 5 15
293 HULA-B 1340 1350 10 15
155 COOK-A 70 75 5 15
172 COOK-A 255 265 10 15
177 COOK-A 280 295 15 15
184 COOK-A 330 375 45 15
191 COOK-A 390 405 15 15
200 COOK-A 480 495 15 15
206 COOK-A 520 535 15 15
218 COOK-A 570 580 10 15
226 COOK-A 645 650 5 15
78 COOK-A 794 808 14 20
84 COOK-A 860 876 16 20
96 COOK-A 962 975 13 20
262 COOK-A 1010 1025 15 15
277 COOK-A 1170 1215 45 15
124 COOK-A 1254 1270 16 20
287 COOK-A 1285 1290 5 15
134 COOK-A 1356 1367 11 20
173 COOK-B 260 730 470 15
241 COOK-B 825 870 45 15
247 COOK-B 885 930 45 15
283 COOK-B 1235 1285 50 15
136 COOK-B 1361 1373 12 20
152 INDI-A 53 63 10 15
157 INDI-A 80 90 10 15
158 INDI-A 105 110 5 15
163 INDI-A 135 140 5 15
15 INDI-A 166 182 16 20
167 INDI-A 210 220 10 15
168 INDI-A 235 255 20 15
179 INDI-A 285 290 5 15
181 INDI-A 330 335 5 15
186 INDI-A 350 360 10 15
183 INDI-A 375 395 20 15
38 INDI-A 417 430 13 20
197 INDI-A 450 470 20 15
203 INDI-A 505 510 5 15
208 INDI-A 540 555 15 15
212 INDI-A 570 580 10 15
231 INDI-A 690 695 5 15
230 INDI-A 710 730 20 15
239 INDI-A 800 810 10 15
242 INDI-A 830 850 20 15
245 INDI-A 870 890 20 15
250 INDI-A 905 910 5 15
252 INDI-A 935 955 20 15
257 INDI-A 990 1010 20 15

264 INDI-A 1040 1050 10 15
265 INDI-A 1065 1075 10 15
266 INDI-A 1090 1095 5 15
278 INDI-A 1180 1185 5 15
282 INDI-A 1208 1218 10 15
279 INDI-A 1233 1243 10 15
288 INDI-A 1280 1290 10 15
130 INDI-A 1313 1328 15 20
294 INDI-A 1350 1370 20 15
298 INDI-A 1385 1390 5 15
140 INDI-A 1417 1431 14 20
148 BOSS-A 23 38 15 15
7 BOSS-A 92 108 16 20
162 BOSS-A 135 150 15 15
175 BOSS-A 270 290 20 15
28 BOSS-A 319 335 16 20
189 BOSS-A 375 390 15 15
41 BOSS-A 449 463 14 20
48 BOSS-A 499 512 13 20
55 BOSS-A 562 577 15 20
61 BOSS-A 626 643 17 20
223 BOSS-A 658 668 10 15
234 BOSS-A 715 910 195 15
94 BOSS-A 950 963 13 20
104 BOSS-A 1050 1066 16 20
114 BOSS-A 1161 1174 13 20
121 BOSS-A 1214 1229 15 20
131 BOSS-A 1313 1330 17 20
299 BOSS-A 1380 1400 20 15
142 BOSS-A 1437 1453 16 20
145 BOSS-B 5 10 5 15
9 BOSS-B 112 128 16 20
180 BOSS-B 320 330 10 15
188 BOSS-B 371 400 29 15
194 BOSS-B 420 430 10 15
201 BOSS-B 485 500 15 15
54 BOSS-B 556 567 11 20
58 BOSS-B 600 615 15 20
66 BOSS-B 656 672 16 20
236 BOSS-B 750 770 20 15
246 BOSS-B 870 880 10 15
259 BOSS-B 990 1650 660 15
1 LION-A 20 35 15 20
159 LION-A 90 110 20 15
19 LION-A 203 216 13 20
22 LION-A 251 262 11 20
26 LION-A 300 314 14 20
31 LION-A 351 367 16 20
42 LION-A 453 468 15 20
205 LION-A 515 535 20 15
219 LION-A 570 590 20 15
217 LION-A 605 625 20 15
65 LION-A 645 658 13 20
72 LION-A 745 762 17 20
85 LION-A 869 882 13 20
91 LION-A 929 944 15 20
255 LION-A 960 1020 60 15
105 LION-A 1060 1073 13 20
111 LION-A 1136 1153 17 20
116 LION-A 1189 1203 14 20
126 LION-A 1267 1282 15 20
300 LION-A 1395 1400 5 15
141 LION-A 1430 1444 14 20
154 LION-B 60 360 300 15
196 LION-B 440 460 20 15
209 LION-B 560 570 10 15
222 LION-B 620 650 30 15
83 LION-B 847 861 14 20
102 LION-B 1038 1050 12 20
263 LION-B 1065 1085 20 15
271 LION-B 1110 1195 85 15
122 LION-B 1239 1254 15 20
296 LION-B 1365 1415 50 15
302 LION-B 1430 1450 20 15
13 GUAM-A 152 167 15 20
174 GUAM-A 265 285 20 15
33 GUAM-A 367 381 14 20
44 GUAM-A 466 476 10 20
211 GUAM-A 560 570 10 15
60 GUAM-A 623 637 14 20
89 GUAM-A 915 931 16 20
101 GUAM-A 1031 1044 13 20
108 GUAM-A 1095 1109 14 20
117 GUAM-A 1194 1204 10 20
286 GUAM-A 1260 1280 20 15
291 GUAM-A 1320 1485 165 15

193 GUAM-B 405 450 45 15
110 GUAM-B 1128 1141 13 20
2 PIKE-A 27 38 11 20
10 PIKE-A 125 141 16 20
27 PIKE-A 306 321 15 20
53 PIKE-A 548 560 12 20
57 PIKE-A 599 611 12 20
64 PIKE-A 644 658 14 20
69 PIKE-A 695 710 15 20
75 PIKE-A 758 774 16 20
112 PIKE-A 1152 1168 16 20
303 PIKE-A 1410 1430 20 15
147 REEF-A 10 30 20 15
5 REEF-A 67 81 14 20
43 REEF-A 460 476 16 20
67 REEF-A 684 698 14 20
80 REEF-A 818 834 16 20
90 REEF-A 921 932 11 20
98 REEF-A 975 993 18 20
268 REEF-A 1080 1100 20 15
119 REEF-A 1211 1227 16 20
132 REEF-A 1320 1332 12 20
138 REEF-A 1394 1409 15 20
305 REEF-A 1425 1435 10 15

Day 6

6 POGO-A 70 86 16 20
10 POGO-A 107 124 17 20
159 POGO-A 139 144 5 15
15 POGO-A 172 186 14 20
19 POGO-A 210 226 16 20
24 POGO-A 254 270 16 20
29 POGO-A 298 314 16 20
35 POGO-A 354 370 16 20
39 POGO-A 401 414 13 20
44 POGO-A 454 471 17 20
49 POGO-A 500 516 16 20
55 POGO-A 555 572 17 20
64 POGO-A 620 636 16 20
68 POGO-A 682 696 14 20
73 POGO-A 719 735 16 20
79 POGO-A 812 826 14 20
82 POGO-A 863 876 13 20
240 POGO-A 900 915 15 15
95 POGO-A 988 999 11 20
100 POGO-A 1020 1033 13 20
251 POGO-A 1048 1053 5 15
104 POGO-A 1101 1116 15 20
257 POGO-A 1131 1136 5 15
111 POGO-A 1174 1187 13 20
117 POGO-A 1218 1232 14 20
277 POGO-A 1250 1275 25 15
126 POGO-A 1310 1326 16 20
285 POGO-A 1341 1356 15 15
135 POGO-A 1379 1395 16 20
297 POGO-A 1415 1425 10 15
5 POGO-B 53 69 16 20
9 POGO-B 102 116 14 20
14 POGO-B 153 170 17 20
162 POGO-B 185 200 15 15
21 POGO-B 225 236 11 20
173 POGO-B 270 280 10 15
33 POGO-B 324 336 12 20
177 POGO-B 351 356 5 15
186 POGO-B 385 395 10 15
41 POGO-B 422 436 14 20
192 POGO-B 451 466 15 15
198 POGO-B 485 495 10 15
52 POGO-B 520 534 14 20
57 POGO-B 581 593 12 20
62 POGO-B 617 632 15 20
217 POGO-B 665 675 10 15
70 POGO-B 715 729 14 20
224 POGO-B 770 805 35 15
229 POGO-B 840 855 15 15
83 POGO-B 882 898 16 20
89 POGO-B 924 938 14 20
244 POGO-B 960 975 15 15
97 POGO-B 1007 1023 16 20
254 POGO-B 1045 1055 10 15
103 POGO-B 1086 1099 13 20
107 POGO-B 1119 1132 13 20
266 POGO-B 1150 1165 15 15
264 POGO-B 1180 1195 15 15
120 POGO-B 1234 1246 12 20
122 POGO-B 1279 1295 16 20
129 POGO-B 1317 1333 16 20
134 POGO-B 1353 1367 14 20
142 POGO-B 1413 1427 14 20
148 POGO-C 40 75 35 15
156 POGO-C 90 110 20 15
161 POGO-C 130 150 20 15
18 POGO-C 199 213 14 20
30 POGO-C 298 310 12 20
179 POGO-C 330 340 10 15
181 POGO-C 360 375 15 15
38 POGO-C 399 415 16 20
188 POGO-C 430 445 15 15
197 POGO-C 480 495 15 15
51 POGO-C 517 533 16 20
58 POGO-C 588 602 14 20
211 POGO-C 617 632 15 15
218 POGO-C 675 710 35 15
76 POGO-C 782 797 15 20
231 POGO-C 845 865 20 15
84 POGO-C 888 899 11 20
239 POGO-C 914 929 15 15
247 POGO-C 985 995 10 15
110 POGO-C 1126 1137 11 20
267 POGO-C 1160 1170 10 15
270 POGO-C 1185 1195 10 15
118 POGO-C 1222 1234 12 20
128 POGO-C 1312 1325 13 20
281 POGO-C 1340 1350 10 15
286 POGO-C 1365 1380 15 15
141 POGO-C 1413 1428 15 20
147 HULA-A 30 50 20 15
158 HULA-A 90 95 5 15
160 HULA-A 120 125 5 15
16 HULA-A 174 190 16 20
167 HULA-A 230 240 10 15
25 HULA-A 275 290 15 20
174 HULA-A 310 330 20 15
36 HULA-A 359 373 14 20
40 HULA-A 408 423 15 20
191 HULA-A 450 465 15 15
195 HULA-A 480 490 10 15
202 HULA-A 520 525 5 15
206 HULA-A 555 595 40 15
215 HULA-A 645 660 15 15
220 HULA-A 700 710 10 15
225 HULA-A 780 790 10 15
230 HULA-A 840 855 15 15
235 HULA-A 870 880 10 15
93 HULA-A 948 963 15 20
250 HULA-A 990 1010 20 15
102 HULA-A 1048 1063 15 20
255 HULA-A 1078 1083 5 15
261 HULA-A 1110 1125 15 15
263 HULA-A 1140 1160 20 15
114 HULA-A 1200 1215 15 20
276 HULA-A 1245 1250 5 15
131 HULA-A 1328 1342 14 20
289 HULA-A 1375 1385 10 15
290 HULA-A 1400 1405 5 15
144 HULA-A 1429 1444 15 20
146 HULA-B 30 45 15 15
151 HULA-B 60 85 25 15
163 HULA-B 190 200 10 15
165 HULA-B 220 270 50 15
27 HULA-B 293 309 16 20
182 HULA-B 360 370 10 15
187 HULA-B 405 430 25 15
45 HULA-B 457 473 16 20
194 HULA-B 488 493 5 15
200 HULA-B 510 555 45 15
212 HULA-B 600 645 45 15
232 HULA-B 850 885 35 15
90 HULA-B 927 941 14 20
241 HULA-B 956 971 15 15
243 HULA-B 986 991 5 15
101 HULA-B 1037 1053 16 20
258 HULA-B 1080 1125 45 15
271 HULA-B 1200 1215 15 15
278 HULA-B 1275 1315 40 15

284 HULA-B 1335 1345 10 15
296 HULA-B 1410 1445 35 15
1 COOK-A 11 25 14 20
149 COOK-A 45 80 35 15
166 COOK-A 225 230 5 15
168 COOK-A 245 265 20 15
34 COOK-A 327 343 16 20
183 COOK-A 375 405 30 15
193 COOK-A 455 465 10 15
196 COOK-A 480 490 10 15
203 COOK-A 535 560 25 15
205 COOK-A 575 595 20 15
222 COOK-A 750 755 5 15
233 COOK-A 850 880 30 15
237 COOK-A 895 935 40 15
246 COOK-A 975 1020 45 15
259 COOK-A 1095 1110 15 15
268 COOK-A 1170 1185 15 15
119 COOK-A 1223 1239 16 20
130 COOK-A 1320 1365 45 20
138 COOK-A 1392 1406 14 20
293 COOK-A 1421 1436 15 15
2 COOK-B 17 32 15 20
31 COOK-B 315 329 14 20
184 COOK-B 380 395 15 15
199 COOK-B 510 530 20 15
221 COOK-B 715 900 185 15
91 COOK-B 933 948 15 20
274 COOK-B 1225 1270 45 15
291 COOK-B 1390 1645 255 15
145 INDI-A 20 25 5 15
4 INDI-A 52 66 14 20
157 INDI-A 81 91 10 15
12 INDI-A 138 152 14 20
164 INDI-A 210 230 20 15
171 INDI-A 265 285 20 15
170 INDI-A 300 305 5 15
175 INDI-A 320 330 10 15
180 INDI-A 350 400 50 15
185 INDI-A 415 430 15 15
190 INDI-A 450 470 20 15
201 INDI-A 515 535 20 15
204 INDI-A 550 560 10 15
207 INDI-A 575 585 10 15
214 INDI-A 615 620 5 15
210 INDI-A 635 640 5 15
219 INDI-A 690 710 20 15
223 INDI-A 760 795 35 15
227 INDI-A 810 850 40 15
228 INDI-A 865 870 5 15
234 INDI-A 885 905 20 15
242 INDI-A 935 955 20 15
245 INDI-A 975 980 5 15
249 INDI-A 995 1015 20 15
253 INDI-A 1040 1050 10 15
256 INDI-A 1080 1100 20 15
262 INDI-A 1120 1145 25 15
269 INDI-A 1170 1215 45 15
273 INDI-A 1230 1240 10 15
123 INDI-A 1282 1298 16 20
282 INDI-A 1313 1318 5 15
280 INDI-A 1333 1338 5 15
288 INDI-A 1375 1425 50 15
295 INDI-A 1440 1460 20 15
152 BOSS-A 49 113 64 15
11 BOSS-A 135 150 15 20
26 BOSS-A 290 304 14 20
172 BOSS-A 319 339 20 15
189 BOSS-A 405 450 45 15
46 BOSS-A 481 495 14 20
54 BOSS-A 529 544 15 20
56 BOSS-A 580 591 11 20
65 BOSS-A 628 643 15 20
216 BOSS-A 660 670 10 15
226 BOSS-A 800 810 10 15
88 BOSS-A 920 931 11 20
238 BOSS-A 946 956 10 15
99 BOSS-A 1019 1036 17 20
252 BOSS-A 1051 1071 20 15
108 BOSS-A 1121 1135 14 20
265 BOSS-A 1150 1155 5 15
113 BOSS-A 1191 1205 14 20
275 BOSS-A 1240 1250 10 15
279 BOSS-A 1280 1290 10 15
127 BOSS-A 1311 1321 10 20
283 BOSS-A 1336 1346 10 15
140 BOSS-A 1408 1424 16 20
50 BOSS-B 514 526 12 20
208 BOSS-B 570 590 20 15
209 BOSS-B 605 620 15 15
66 BOSS-B 647 663 16 20
236 BOSS-B 885 930 45 15
124 BOSS-B 1288 1298 10 20
137 BOSS-B 1386 1403 17 20
299 BOSS-B 1430 1450 20 15
153 LION-A 52 62 10 15
7 LION-A 89 106 17 20
17 LION-A 188 203 15 20
22 LION-A 233 247 14 20
32 LION-A 323 338 15 20
178 LION-A 353 363 10 15
42 LION-A 424 440 16 20
53 LION-A 527 537 10 20
61 LION-A 615 626 11 20
71 LION-A 715 731 16 20
75 LION-A 756 771 15 20
78 LION-A 805 814 9 20
86 LION-A 897 910 13 20
98 LION-A 1011 1021 10 20
106 LION-A 1108 1124 16 20
116 LION-A 1210 1225 15 20
294 LION-A 1405 1420 15 15
292 LION-A 1435 1445 10 15
150 LION-B 45 480 435 15
213 LION-B 605 625 20 15
80 LION-B 816 831 15 20
87 LION-B 899 913 14 20
96 LION-B 994 1009 15 20
260 LION-B 1110 1190 80 15
272 LION-B 1207 1217 10 15
139 LION-B 1401 1413 12 20
298 LION-B 1430 1440 10 15
155 GUAM-A 60 100 40 15
20 GUAM-A 223 238 15 20
176 GUAM-A 330 350 20 15
37 GUAM-A 397 412 15 20
48 GUAM-A 494 508 14 20
59 GUAM-A 594 609 15 20
94 GUAM-A 987 1000 13 20
109 GUAM-A 1125 1140 15 20
121 GUAM-A 1241 1257 16 20
132 GUAM-A 1344 1355 11 20
287 GUAM-A 1370 1390 20 15
169 GUAM-B 255 730 475 15
248 GUAM-B 985 1390 405 15
8 PIKE-A 96 112 16 20
47 PIKE-A 489 505 16 20
67 PIKE-A 676 689 13 20
74 PIKE-A 732 747 15 20
81 PIKE-A 831 847 16 20
125 PIKE-A 1294 1305 11 20
136 PIKE-A 1385 1399 14 20
3 REEF-A 40 51 11 20
13 REEF-A 138 154 16 20
23 REEF-A 246 260 14 20
43 REEF-A 430 445 15 20
63 REEF-A 619 633 14 20
69 REEF-A 686 700 14 20
77 REEF-A 789 805 16 20
85 REEF-A 891 905 14 20
112 REEF-A 1181 1197 16 20
133 REEF-A 1350 1365 15 20
143 REEF-A 1425 1439 14 20

Day 7

143 POGO-A 15 30 15 15
153 POGO-A 60 75 15 15
155 POGO-A 90 105 15 15
161 POGO-A 125 145 20 15
164 POGO-A 190 200 10 15
168 POGO-A 245 275 30 15
177 POGO-A 325 335 10 15
183 POGO-A 375 390 15 15
191 POGO-A 420 435 15 15
185 POGO-A 450 460 10 15

196 POGO-A 500 545 45 15
51 POGO-A 572 588 16 20
208 POGO-A 603 618 15 15
60 POGO-A 648 663 15 20
213 POGO-A 678 688 10 15
72 POGO-A 721 732 11 20
227 POGO-A 780 785 5 15
231 POGO-A 835 850 15 15
85 POGO-A 878 895 17 20
88 POGO-A 921 931 10 20
242 POGO-A 946 961 15 15
98 POGO-A 992 1004 12 20
251 POGO-A 1020 1035 15 15
254 POGO-A 1050 1060 10 15
107 POGO-A 1091 1104 13 20
260 POGO-A 1119 1129 10 15
268 POGO-A 1155 1165 10 15
116 POGO-A 1195 1212 17 20
119 POGO-A 1249 1264 15 20
126 POGO-A 1297 1313 16 20
282 POGO-A 1330 1342 12 15
134 POGO-A 1368 1384 16 20
295 POGO-A 1410 1420 10 15
291 POGO-A 1435 1445 10 15
147 POGO-B 35 70 35 15
158 POGO-B 115 130 15 15
160 POGO-B 145 160 15 15
163 POGO-B 175 190 15 15
170 POGO-B 265 275 10 15
179 POGO-B 330 375 45 15
190 POGO-B 405 450 45 15
193 POGO-B 465 510 45 15
47 POGO-B 550 561 11 20
52 POGO-B 590 606 16 20
206 POGO-B 621 636 15 15
66 POGO-B 674 690 16 20
218 POGO-B 705 720 15 15
73 POGO-B 745 759 14 20
80 POGO-B 821 831 10 20
82 POGO-B 851 867 16 20
235 POGO-B 895 910 15 15
92 POGO-B 938 952 14 20
99 POGO-B 994 1010 16 20
247 POGO-B 1025 1035 10 15
106 POGO-B 1081 1097 16 20
110 POGO-B 1118 1131 13 20
112 POGO-B 1159 1172 13 20
275 POGO-B 1215 1255 40 15
124 POGO-B 1289 1304 15 20
132 POGO-B 1348 1363 15 20
140 POGO-B 1400 1415 15 20
157 POGO-C 90 135 45 15
162 POGO-C 165 180 15 15
167 POGO-C 240 255 15 15
173 POGO-C 285 330 45 15
184 POGO-C 375 405 30 15
194 POGO-C 480 490 10 15
200 POGO-C 530 555 25 15
53 POGO-C 598 614 16 20
62 POGO-C 651 663 12 20
68 POGO-C 691 707 16 20
74 POGO-C 751 766 15 20
232 POGO-C 840 860 20 15
234 POGO-C 885 930 45 15
244 POGO-C 975 1020 45 15
108 POGO-C 1094 1111 17 20
267 POGO-C 1140 1150 10 15
114 POGO-C 1182 1196 14 20
118 POGO-C 1230 1244 14 20
121 POGO-C 1269 1284 15 20
128 POGO-C 1318 1331 13 20
281 POGO-C 1346 1361 15 15
138 POGO-C 1381 1394 13 20
292 POGO-C 1409 1424 15 15
145 HULA-A 30 50 20 15
149 HULA-A 65 75 10 15
156 HULA-A 90 100 10 15
159 HULA-A 120 130 10 15
12 HULA-A 161 177 16 20
165 HULA-A 210 215 5 15
22 HULA-A 262 277 15 20
171 HULA-A 292 297 5 15
27 HULA-A 317 329 12 20
32 HULA-A 359 374 15 20
181 HULA-A 389 404 15 15
38 HULA-A 443 455 12 20
195 HULA-A 480 495 15 15
201 HULA-A 530 580 50 15
202 HULA-A 595 605 10 15
209 HULA-A 630 635 5 15
212 HULA-A 660 665 5 15
216 HULA-A 690 695 5 15
220 HULA-A 710 720 10 15
75 HULA-A 758 771 13 20
79 HULA-A 802 816 14 20
95 HULA-A 966 980 14 20
239 HULA-A 995 1005 10 15
102 HULA-A 1034 1049 15 20
249 HULA-A 1064 1079 15 15
109 HULA-A 1112 1123 11 20
265 HULA-A 1138 1148 10 15
269 HULA-A 1170 1175 5 15
266 HULA-A 1190 1195 5 15
273 HULA-A 1210 1255 45 15
280 HULA-A 1285 1295 10 15
277 HULA-A 1310 1320 10 15
284 HULA-A 1345 1360 15 15
139 HULA-A 1398 1414 16 20
9 HULA-B 136 151 15 20
24 HULA-B 265 280 15 20
36 HULA-B 426 442 16 20
197 HULA-B 510 530 20 15
199 HULA-B 545 560 15 15
203 HULA-B 575 615 40 15
211 HULA-B 630 645 15 15
214 HULA-B 670 695 25 15
224 HULA-B 765 800 35 15
230 HULA-B 825 870 45 15
91 HULA-B 934 949 15 20
100 HULA-B 1008 1025 17 20
248 HULA-B 1040 1060 20 15
258 HULA-B 1090 1105 15 15
263 HULA-B 1125 1150 25 15
270 HULA-B 1170 1185 15 15
122 HULA-B 1265 1280 15 20
278 HULA-B 1295 1335 40 15
293 HULA-B 1385 1420 35 15
2 COOK-A 49 63 14 20
18 COOK-A 217 232 15 20
25 COOK-A 271 284 13 20
29 COOK-A 350 362 12 20
180 COOK-A 377 382 5 15
48 COOK-A 558 574 16 20
223 COOK-A 760 765 5 15
87 COOK-A 904 920 16 20
240 COOK-A 940 955 15 15
245 COOK-A 985 995 10 15
104 COOK-A 1071 1085 14 20
255 COOK-A 1100 1115 15 15
264 COOK-A 1130 1145 15 15
276 COOK-A 1250 1275 25 15
125 COOK-A 1295 1309 14 20
285 COOK-A 1350 1365 15 15
141 COOK-A 1415 1429 14 20
207 COOK-B 570 595 25 15
225 COOK-B 765 800 35 15
241 COOK-B 940 970 30 15
259 COOK-B 1095 1140 45 15
142 COOK-B 1424 1438 14 20
150 INDI-A 51 61 10 15
154 INDI-A 80 90 10 15
11 INDI-A 153 162 9 20
17 INDI-A 215 265 50 20
174 INDI-A 300 310 10 15
178 INDI-A 330 350 20 15
182 INDI-A 370 375 5 15
188 INDI-A 400 425 25 15
192 INDI-A 450 470 20 15
43 INDI-A 500 515 15 20
204 INDI-A 560 570 10 15
56 INDI-A 601 615 14 20
215 INDI-A 675 680 5 15
70 INDI-A 710 725 15 20
228 INDI-A 800 810 10 15
229 INDI-A 825 830 5 15
233 INDI-A 870 890 20 15
236 INDI-A 905 910 5 15
94 INDI-A 965 978 13 20
243 INDI-A 993 1003 10 15
252 INDI-A 1035 1040 5 15
253 INDI-A 1055 1065 10 15
256 INDI-A 1080 1100 20 15

262 INDI-A 1115 1190 75 15
274 INDI-A 1210 1225 15 15
120 INDI-A 1252 1268 16 20
279 INDI-A 1283 1293 10 15
287 INDI-A 1350 1370 20 15
286 INDI-A 1385 1395 10 15
296 INDI-A 1415 1435 20 15
1 BOSS-A 30 46 16 20
8 BOSS-A 130 145 15 20
21 BOSS-A 260 273 13 20
172 BOSS-A 288 293 5 15
176 BOSS-A 320 330 10 15
31 BOSS-A 359 375 16 20
187 BOSS-A 390 405 15 15
189 BOSS-A 420 425 5 15
40 BOSS-A 463 475 12 20
44 BOSS-A 502 512 10 20
46 BOSS-A 534 548 14 20
54 BOSS-A 599 614 15 20
210 BOSS-A 630 640 10 15
64 BOSS-A 667 681 14 20
219 BOSS-A 696 701 5 15
221 BOSS-A 720 730 10 15
222 BOSS-A 750 765 15 15
237 BOSS-A 930 940 10 15
96 BOSS-A 980 985 5 20
250 BOSS-A 1020 1040 20 15
261 BOSS-A 1110 1125 15 15
271 BOSS-A 1205 1225 20 15
283 BOSS-A 1345 1350 5 15
290 BOSS-A 1380 1400 20 15
289 BOSS-A 1415 1435 20 15
151 BOSS-B 60 360 300 15
45 BOSS-B 512 527 15 20
49 BOSS-B 560 574 14 20
55 BOSS-B 600 617 17 20
205 BOSS-B 632 652 20 15
69 BOSS-B 700 716 16 20
97 BOSS-B 989 1005 16 20
130 BOSS-B 1323 1336 13 20
137 BOSS-B 1380 1395 15 20
4 LION-A 59 75 16 20
7 LION-A 124 137 13 20
13 LION-A 167 178 11 20
19 LION-A 220 235 15 20
166 LION-A 250 270 20 15
26 LION-A 294 309 15 20
175 LION-A 324 329 5 15
33 LION-A 361 374 13 20
186 LION-A 390 395 5 15
198 LION-A 515 535 20 15
65 LION-A 670 679 9 20
217 LION-A 694 714 20 15
77 LION-A 786 802 16 20
81 LION-A 834 845 11 20
90 LION-A 929 943 14 20
238 LION-A 958 978 20 15
101 LION-A 1027 1040 13 20
257 LION-A 1080 1100 20 15
272 LION-A 1207 1217 10 15
123 LION-A 1285 1297 12 20
131 LION-A 1334 1349 15 20
135 LION-A 1372 1382 10 20
297 LION-A 1423 1433 10 15
146 LION-B 30 53 23 15
23 LION-B 263 278 15 20
28 LION-B 321 331 10 20
34 LION-B 395 412 17 20
42 LION-B 497 510 13 20
67 LION-B 685 700 15 20
226 LION-B 780 840 60 15
89 LION-B 929 943 14 20
246 LION-B 990 1010 20 15
105 LION-B 1080 1095 15 20
113 LION-B 1180 1197 17 20
288 LION-B 1365 1370 5 15
294 LION-B 1410 1430 20 15
152 GUAM-A 60 90 30 15
15 GUAM-A 192 208 16 20
20 GUAM-A 250 305 55 20
30 GUAM-A 350 364 14 20
37 GUAM-A 428 441 13 20
41 GUAM-A 466 478 12 20
50 GUAM-A 565 581 16 20
58 GUAM-A 615 629 14 20
83 GUAM-A 855 869 14 20
86 GUAM-A 890 935 45 20
93 GUAM-A 956 971 15 20
103 GUAM-A 1059 1072 13 20
117 GUAM-A 1213 1228 15 20
127 GUAM-A 1314 1328 14 20
169 GUAM-B 265 285 20 15
39 GUAM-B 449 459 10 20
5 PIKE-A 68 84 16 20
16 PIKE-A 195 208 13 20
57 PIKE-A 610 624 14 20
63 PIKE-A 659 673 14 20
78 PIKE-A 802 818 16 20
115 PIKE-A 1192 1208 16 20
129 PIKE-A 1320 1365 45 20
144 REEF-A 25 50 25 15
6 REEF-A 109 126 17 20
35 REEF-A 402 413 11 20
59 REEF-A 620 632 12 20
71 REEF-A 716 730 14 20
76 REEF-A 762 775 13 20
84 REEF-A 862 877 15 20
111 REEF-A 1151 1166 15 20
133 REEF-A 1359 1372 13 20

Bibliography

1. Arbabi, Mansur and John A. Garate. "Interactive Real Time Scheduling and Control." *Proceedings of the 1985 Summer Simulation Conference*. 271 –277. 1985.
2. Baker, Bruce N. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, 1974.
3. Blanton Jr., Joe L. and Roger L. Wainwright. "Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms." *Proceedings of 5th International Conference on Genetic Algorithms*, edited by S. Forrest. San Mateo, CA: Morgan Kauffman, 1993.
4. Bruns, Ralf. "Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling." *Proceedings of 5th International Conference on Genetic Algorithms*, edited by S. Forrest. San Mateo, CA: Morgan Kauffman, 1993.
5. Davis, Lawerence. *The Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.
6. Garey, Michael R. and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman, 1979.
7. Goldberg, David. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley, 1989.
8. Gooley, Capt Timothy. *Automating the Satellite Range Scheduling Process*. MS thesis, AFIT/GOR/ENS/93M-06, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1993. (AAK-1926).
9. Grefenstette, John, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. "Genetic Algorithms for the Traveling Salesman Problem." *Proceedings of an International Conference on Genetic Algorithms and their Applications*, edited by John Grefenstette. Hillsdale, NJ: Lawrence Erlbaum Associates, 1985.
10. Holland, John. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975. reprinted by MIT Press, Cambridge, MA, 1992.
11. Kennedy, Capt Dale J. *A Prototype Expert System Advisor for Satellite Support Scheduling*. MS thesis, AFIT/GOR/OS/86D-5, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1986. (AD-A179 425).
12. Lawler, E.L., editor. *The Traveling Salesman Problem*. UK: John Wiley and Sons, 1985.
13. Michalewicz, Zbigniew. *Genetic Algorithms + Data Structure = Evolution Programs*. New York: Springer-Verlag, 1992.
14. Schalck, Capt Stanley Michael. *Automating Satellite Range Scheduling*. MS thesis, AFIT/GSO/ENS/93D-14, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1993. (AD-A273 829).

15. Starkweather, T. and Darryl Whitley. "A Comparison of Genetic Sequencing Operators." *Proceedings of 4th International Conference on Genetic Algorithms*, edited by R.K. Belew and L.B. Booker. Los Altos, CA: Morgan Kaufmann, 1991.

16. Syswerda, Gilbert. *The Handbook of Genetic Algorithms*, chapter 21. New York: Van Nostrand Reinhold, 1991. Schedule Optimization Using Genetic Algorithms.

17. Whitley, Darrell. "GENITOR: a different genetic algorithm." *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*. 118–130. 1988.

18. Whitley, Darrell, Timothy Starkweather, and Daniel Shaner. *The Handbook of Genetic Algorithms*, chapter 22. New York: Van Nostrand Reinhold, 1991. The Traveling Salesman and Sequence Scheduling: Quality Solutions using Genetic Edge Recombination.

19. Whitley, Darryl, Timothy Starkweather, and D'Ann Fuquay. "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator." *Proceedings of 3rd International Conference on Genetic Algorithms*, edited by J.D. Schaffer. Los Altos, CA: Morgan Kaufmann, 1989.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1994	Master's Thesis

4. TITLE AND SUBTITLE
A GENETIC ALGORITHM APPROACH TO AUTOMATING SATELLITE RANGE SCHEDULING

5. FUNDING NUMBERS

6. AUTHOR(S)
Donald A. Parish, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/94M-10

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Satellite range scheduling involves scheduling satellite supports in which a satellite and a specific remote tracking station communicate with each other within a specified time window. As the number of satellite supports continue to increase, more pressure is placed on the current manual system to generate schedules efficiently. Previous research efforts focused on heuristic and mixed-integer programming approaches which may not produce the best results. The objective of this research was to determine if a genetic algorithm approach to automating the generation of 24 hour schedules was competitive with other methods. The goal was to schedule as many supports as possible without conflict. The genetic algorithm approach attempted to find the best priority ordering of support requests, and then used a schedule builder program to build schedules based on simple rules. A schedule was produced for seven days of representative satellite range data with slightly better results compared to earlier results using a mixed-integer programming formulation. Based on the reported results, the genetic algorithm approach presented in this research appears to be a competitive approach for generating 24-hour satellite range schedules.

14. SUBJECT TERMS
Genetic Algorithms, Scheduling, Satellite range scheduling

15. NUMBER OF PAGES
101

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL