SEARCHING FOR PRESCRIPTIVE TREATMENT SCHEDULES WITH A GENETIC ALGORITHM: A TOOL FOR FOREST MANAGEMENT

by

JOHN DEWEY

(Under the Direction of Walter D. Potter)

ABSTRACT

This thesis describes research on the use of a genetic algorithm (GA) to prescribe treatment plans for forest management at the stand level. Forest management refers to making decisions about when and where to intervene in the natural growth of forests to achieve objectives, such as enhancing the visual quality of a stand or maximizing timber yield. A prescription is a schedule of thinning treatments applied to stands over a planning horizon.

When multiple management goals exist treatment prescription becomes a complex multi-objective problem. The effectiveness of a GA depends on selecting an appropriate representation and germane fitness function. These design decisions are reviewed, followed by a series of experiments testing the performance of the GA. Different parameter settings are compared and the GA is contrasted with some other heuristic search methods. The final experiment compares a plan created by the GA to a plan recommended by a human expert.

INDEX WORDS: Genetic algorithms, Decision support systems, NED, Forest management, Prolog, Silviculture, Harvesting schedule, Treatment prescription

SEARCHING FOR PRESCRIPTIVE TREATMENT SCHEDULES WITH A GENETIC ALGORITHM: A TOOL FOR FOREST MANAGEMENT

by

JOHN DEWEY

A.B., University of Georgia, 2003

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2005

SEARCHING FOR PRESCRIPTIVE TREATMENT SCHEDULES WITH A GENETIC ALGORITHM: A TOOL FOR FOREST MANAGEMENT

by

JOHN DEWEY

ACKNOWLEDGEMENTS

I would like to thank my committee members, Don Potter, Donald Nute, and James Brown for their positive influences on my intellectual development at Georgia. Having professors with such a diverse set of interests has helped steer me into interdisciplinary cognitive studies.

Thanks also go to Mike Rauscher, who provided domain expert data for comparison to the GA, and also to the rest of the NED team for their help.

Special thanks go to Fred Maier, who has worked with me on designing and programming the GA. His insights, particularly in the area of representation, have been invaluable.

Thanks to Sarah Hemmings, my friend, roommate, and study partner. I don't know how I would have made it through the last couple of months without her.

Last but not least, thanks to the AI center and the USDA Forest Service for funding my graduate studies, and to my family and friends for their love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

This paper describes research on the use of a genetic algorithm (GA) to prescribe treatment plans for forest management at the stand level. Forest management refers to making decisions about when and where to intervene in the natural growth of forests to achieve objectives, such as enhancing the visual quality of a stand or maximizing timber yield. A prescription is a schedule of thinning treatments applied to stands over a planning horizon.

Broadly speaking, national forests are managed on two levels: forest and project (Holsapple & Whiston, 1996). Forest-level management plans are large scale strategic programs that establish general goals, guidelines, and standards. Forest service personnel typically have flexibility in how they choose to implement a forest-level plan on a more local level (Morrison, 1993). The second level, project-level plans, is tactical and site specific in nature. The prescriptive approach described here is an example of a project-level management plan.

Treatment prescription at the project level is a complex multiple constraint satisfaction problem with a search space that increases exponentially with the length of the plan (that is, the number of years the plan covers divided by the treatment interval). Human experts can not realistically consider every alternative plan and must instead fall back on general rules of thumb acquired from experience. One motivation for using machine learning techniques is to take some of the guesswork and bias out of planning by providing users with concrete recommendations.

Many optimization techniques have been applied to the problem of stand-level optimization. It is not the purpose of this paper to provide a comprehensive review of the many approaches that have been taken to this problem, but examples include the Hooke and Jeeves method (1961), dynamic programming, genetic algorithms, tabu search, and various hybrid methods that incorporate different heuristic methods. Bettinger and Boston (2002) for example found that a combination of a GA and tabu search heuristics got better results for one type of harvesting problem than either achieved alone.

A vast literature exists on GAs and their applications to multiple constraint problems. Feng and Lin (1999) used a GA to design several alternative urban plans for the city of Tanhai. They concluded that the GA found better solutions than urban planning experts, and also provided more alternative plans.

In the forestry domain, Mathers, Sibbald, and Craw (1999) developed a GA to strategically categorize stands within a management unit for different functions with the goal of finding the optimal balance of utility within the unit. Some stands might be designated for timber production, others for recreational use or other functions. Ducheyne, De Wulf, and Baets (2001) also used a GA to categorize stands for different purposes, with the goal of optimizing management units along both economic and visual dimensions.

Others have applied GAs to spatially constrained harvest scheduling problems. Mullen and Butler (1999) designed a GA that output the order in which stands should be harvested to optimize timber yields. In a similar project, Hughell and Roise (1997) also used a GA to develop treatment schedules for harvesting. They made use of a simulation model for the endangered red-cockaded woodpecker to balance timber objectives with an ecological concern in an uncertain environment. The output of this GA is a plan that specifies when to harvest from each stand in a management unit. The authors note that although more traditional linear programming techniques are capable of solving this type of optimization problem, scheduling problems with large numbers of decision variables may take a prohibitive amount of CPU time. One of the strengths of GA heuristics is their ability to produce sets of near optimal solutions in a timely manner.

Some researchers have applied different programming strategies to the problem of treatment prescription. Bettinger and Graetz (2004) used dynamic programming techniques to reach stand density targets for forests in the Blue Mountains of eastern Oregon. Their method pools individual solutions for stands with characteristics in common into a generalized consensus opinion that applies to all stands of that type. This is meant to loosely emulate the process by which human experts decide how to treat stands based on experience with similar stands. Bettinger and Graetz's research is similar to the present study in that it focuses on stand-level optimization, where stand-level goals are defined in terms of basic tree-level data such as basal area and trees per acre.

The approach described here differs from similar projects in the literature mainly because a GA is used to develop plans with a high degree of specificity. A framework has been developed that allows the GA to evolve highly specific treatment recommendations at the project-level for a variety of management goals, timber and non-timber (visual or ecological goals) alike. The approach described here does more than lay out a high-level strategy for forest management, it will tell the forester exactly when, where, and how to harvest each stand in the management unit to maximize the number of management goals that can be satisfied in the specified time period. The ultimate measure of the project's success will be how well it performs compared to a human expert, as well as to other machine learning techniques such as simulated annealing and hill-climbing.

1.2 GENETIC ALGORITHMS

Genetic algorithms are an exciting area of research in the field of computational intelligence. As the name suggests, the problem solving mechanism of GAs is loosely modeled after biological evolution. The basic idea is to evolve a solution, represented as a string or similar data structure, from a randomly generated population of candidate solutions using selective reproductive pressures and genetic operators that introduce new candidates into the population.

Evolution in nature depends on natural selection and sexual reproduction. Natural selection determines which individuals survive to reproduce, while sexual reproduction mixes the genetic material from survivors into novel combinations. These biological processes have their analogs in the selection and crossover operations of the GA. Random mutations are also modeled in the GA.

First a population of “chromosomes” is created. Each chromosome represents a candidate solution to the problem at hand. Chromosomes are the basic objects in the GA and are often coded as lists of binary numbers. In the current study, chromosomes are lists of real numbers between zero and one which define a series of forest thinning treatments over many years. Each chromosome string in the population is evaluated by a selection procedure that selects the best, or most “fit”, solutions according to criteria set by a fitness function. The high ranking individuals exchange string information through crossover and produce a new generation of “offspring”. A small percentage of string positions in an offspring may also mutate randomly.

The search power of GAs is attributable to a characteristic called implicit parallelism, which allows large search spaces to be effectively sampled with relatively few computations (Holland, 1992). This is possible because each string in a population belongs to many different regions of the search space simultaneously. For example, the string (1 1 0 0 1) belongs to the region consisting of strings that begin with "1" (1 * * * *), strings with a "0" and a "1" in the fourth and fifth positions (* * * 0 1), and many more. These are examples of *schemata*, sets of strings with common elements in particular positions. When the GA evaluates and assigns fitness to a string, it implicitly samples every schema expressed in the string simultaneously, or in parallel. The *schema theorem* shows that schemata that code for advantageous features become prevalent in GA populations in the same manner as alleles that increase the fitness of biological organisms dominate their gene pools. Initially the schemata that are selected tend to be small, or low order. The building block hypothesis predicts that as evolution takes its course, the most fit small schemata conglomerate into larger blocks (Smith, 1994). Under ideal (and not always realistic) circumstances, the best solution found by the GA can be viewed as the end point of this process: a large, very fit schema which is decomposable into component building blocks representing important dimensions of the problem.

Genetic algorithms are not function optimizers (De Jong, 1993). That is, they are not guaranteed to find the optimal solution in a search space. Therefore it is inappropriate to apply GAs to problems with small search spaces that could be exhaustively searched by algorithmic methods with a reasonable amount of time and computer memory. The strength of the GA approach is that they can often find near optimal solutions to difficult solutions in large spaces, provided a suitable representation and fitness function are specified (Goldberg, 1989). Because

GAs do not compare every possible option, even very large problem spaces can often be searched in a reasonable amount of time.

Another advantage to GAs is that they operate blindly, without preconceived notions about which characteristics of a forest stand might be relevant to the satisfaction of a management goal. All that matters to the GA is how good a treatment plan is, that is, how well the management goals are satisfied in simulation when a particular treatment plan is followed. It is hoped that this equal-opportunity approach will enable the GA to find creative solutions to management problems that might not occur to human experts.

There are many variations on genetic algorithms, and many parameters that can be adjusted that may result in differential levels of performance. The most important aspects of GA design by far however are the representation of the chromosomes and the choice of fitness function. These are the main determining factors in whether or not the GA will function as intended.

Treatment prescription for forested stands is a complex and multidimensional problem. To effectively manage a set of stands, forest managers need to decide what treatments to apply to which stands, and when. Plans that project many years into the future become complicated very quickly, especially when multiple goals are involved. Simulating every alternative (an exhaustive search) is not practical, nor is there any specific method used by human experts to recommend prescriptions. Representing treatments on the other hand is fairly straightforward, because the primary tool of forest managers is forest thinning, and there is also sufficient information available to define useful fitness functions. These issues are discussed in detail in later sections. Thus all the ingredients for a GA approach to the problem are in place.

1.3 NED-2

The current research on the GA for treatment scheduling began as an offshoot of the Northeast Decision Model (NED) project at the University of Georgia. NED-2 is a decision support system (DSS) that provides users with a planning environment for forest management and expert system components for evaluating and comparing plans within a management unit (Nute et al., 2004). A management unit is simply the area of forest under consideration. Management units are divided into stands, which are sections of forest with some uniform characteristics. A plan is a schedule of treatments, such as planting or row thinning, applied to individual stands over predetermined intervals.

NED-2 has been developed with a modular programming philosophy which makes adding new components fairly straightforward. While the GA is being developed using concepts and treatment definitions borrowed from NED-2, it is intended to be usable as a standalone utility. If the approach proves to be effective and practical, it may eventually be incorporated into NED-2 as a new module.

1.4 FVS

Like NED-2, the GA uses the Forest Vegetation Simulator (FVS), a growth and yield simulator produced by the USDA Forest Service, to simulate treatment plans. FVS needs two input files to run a simulation. The parameters describing treatment plans are encoded in a keyword file (*.key). The other input file contains tree inventory data gathered by observation or generated from the NED-2 databases, and is called during execution of the keyword file.

FVS is capable of creating several different output files. The GA only needs data from the optional tree inventory data output file to measure fitness. These data are written to files with the *.out extension. All simulations were run using the southeastern (SN) variant of FVS.

CHAPTER 2

DESIGN OF THE GA

2.1 REPRESENTATION

The simplest representation possible in a GA is a binary string. In many cases candidate solutions to multiple constraint problems are reducible to a list of Boolean values, one or zero for each variable where one indicates the presence of a feature and zero its absence. In addition to being parsimonious there are practical advantages to using a binary alphabet. Mutation is as simple as flipping a bit, and crossover bears a closer resemblance to its counterpart in nature compared to crossover operations in non-binary GAs (De Jong as cited in Deb, 2001). Most importantly, there is reason to believe that binary-coded GAs process schemata more efficiently than other GAs. Goldberg (1991) argued that based on the total number of schemata searched per generation, binary alphabets get the maximum number of schemata per bit of information.

Binary-coded GAs may also require smaller initial populations than other GAs. Reeves (1993) estimated the minimum population size needed by various representations for every solution in the search space to be reachable by crossover alone. For this condition to be met, every member of the alphabet must be represented at every allele position by some member of the population. This ensures that repetitive crossovers will eventually create every possible solution in the search space. The larger the size of the alphabet, the more storage and computer time will be necessary to satisfy the condition. Because binary-coded GAs have only two values in their alphabets, they require smaller populations and thus less storage and computer time to explore their search spaces.

Unfortunately, binary-coded GAs run into difficulties with multi-objective problems in continuous search spaces. One is the difficulty of achieving an arbitrary degree of precision in assigning parameter values. The higher the desired precision is, the longer the strings must be, and the longer the strings become, the larger the optimum population size becomes (Goldberg, Deb, & Clark, 1992). Another problem is that strings that could be considered neighbors in terms of observable features may be very far apart in terms of their representation. One manifestation of this problem is known as a Hamming cliff (Hamming, 1980). The strings (0 1 1 1) and (1 0 0 0), for example, are different in every position, but if interpreted as binary numbers these are neighboring solutions.

In situations where parameters vary over a continuum of values rather than simply being present or absent, parameters can be represented directly in alleles (that is, no string coding) as real numbers. This simplifies representation for certain problems by removing the need to translate real values into discrete, Boolean categories. Real-parameter GAs bring their own unique set of challenges however. In particular, mutation and crossover operators must be reconsidered to ensure that the population is perturbed in a meaningful manner and the search space is explored systematically. It is also not entirely clear what role schemata play in real-parameter GAs. Goldberg (1991) made the case that selection in real-parameter GAs causes the above average solutions to survive. The sub-regions of these solutions constitute a virtual alphabet, meaning that they serve as islands in the search space to which the search is restricted. Effectively, this means the real-parameter GA with the continuous search space is approximated as a discrete search space, and the risk is that the global optimum may lie outside the range of the islands. This concern is ameliorated somewhat by special crossover operations specifically designed for exploring continuous search spaces, an issue to be discussed more later in the paper. For a good overview of real-parameter GAs and their associated genetic operators, see Herrara, Lozano, and Verdegay (1998).

Treatment plans in NED-2 are divided into cycles separated by regular intervals. A typical plan might span 50 years between 2010 and 2060, with a treatment period every 10 years for a total of five cycles. The plan in this example would consist of five treatments, chronologically ordered. Managers may also opt to select no treatment for a cycle and just let the forest grow.

Treatments are applied at the stand level. The main GA program evolves treatment schedules for the management unit one stand at a time, treating stands as independent units (an alternative approach is explored in section 3.5). If multiple stands are selected for optimization, first the GA creates a population of individuals representing plans for the first stand on the list. After evolving the population for the designated number of generations, the GA outputs the stand-level plans it found to an HTML file. This is discussed in greater detail in Chapter Five. The GA repeats this procedure for each stand in the management unit. The prescription for the management unit as a whole is followed by treating each of the component stands according to its own schedule.

There are several predefined treatment types in NED-2. “Thin basal area from above”, “clearcut”, and “shelterwood seed cut” are examples. It is also possible to define custom treatments. Thus one possible representation for a treatment plan in the GA would be a list of chronologically ordered treatment types and cutting efficiencies, as shown in Figure 2.1.

Grow	0.75	Row thinning	0.45	Clearcut	0.75	Thin basal area from above	0.50
Cycle 1		Cycle 2		Cycle 3		Cycle 4	

Figure 2.1: An early representation of treatment plans.

One problem with using this representation for the GA is that small changes in the genotype may result in large changes in phenotype. For example, mutating a "grow" allele into "clearcut" would dramatically affect the status of a forested stand, even if only a single allele is affected. Radcliffe (1992) argues that "genetic operators and analogues of schemata should be defined directly in the space of phenotypes, rather than in the genotype (representation) space". In other words, genetic operations like mutation should operate directly on the features relevant to the problem, rather than on high-level concepts like treatment definitions which only indirectly control the cut.

To address this issue two alternative representations have been devised that represent treatment plans as lists of floating point numbers between zero and one. *Treatments are defined by the diameter at breast height (DBH) of the trees to be cut, and a cutting efficiency which defines the percentage of trees within the range that will actually be cut.* In the first representation (R1) a treatment comprises three alleles: 1) cutting efficiency (CUTEFF), which defines what percentage of trees within the specified range will be cut; 2) a percentage representing the midpoint of the range of DBHs to be cut (STP, a mnemonic for "starting point"); and 3) a percentage used to determine the width of the cutting range (SL, a mnemonic for "slice").

A chromosome with *N* treatment cycles is a list of *N* treatments, where each three-allele triple represents the treatment for a particular cycle: <CUTEFF_1, STP_1, SL_1, …, CUTEFF_n, STP_n, SL_n>. The order of treatments in a chromosome is significant. If a treatment schedule starts in the year 2019 and has 10 year intervals between treatments, the first three alleles in the chromosome code the treatment for 2019, the next three for 2029, and so forth, as shown in Figure 2.2. This representation has a very desirable characteristic: Treatments are coded with just three alleles, which are grouped together. With respect to the building block hypothesis, treatments are obvious candidates for low-order building blocks. And because treatments are only three alleles long, fit treatments are less likely to be destroyed by crossover operations than if they were represented over longer strings. Figure 2.3 shows an example of how treatments are coded in an FVS keyword file.

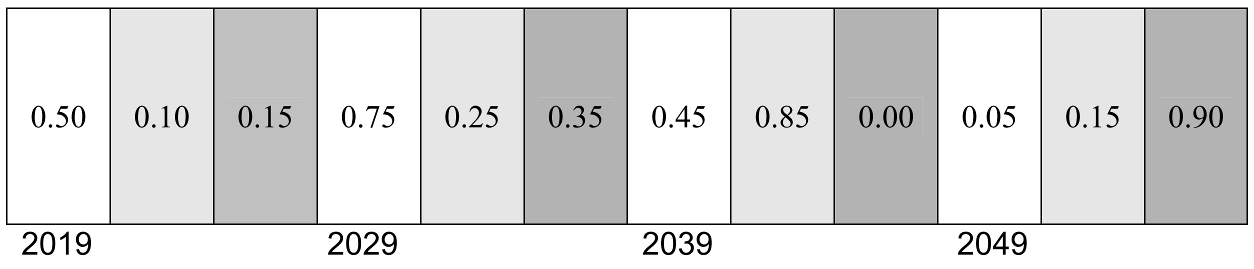


Figure 2.2: The current representation. Treatments are coded over three alleles.

```
COMPUTE         2019
CUTEFF = 0.3
STP = 0.3
SL = 0.4
DBHMIN = DBHDIST(3,1)
DBHMAX = DBHDIST(3,6)
RANGE = DBHMAX - DBHMIN
STPOINT = (STP*RANGE)+DBHMIN
SLICE = (SL * RANGE)
DBHMIN2 = STPOINT - (SLICE/2)
DBHMAX2 = STPOINT + (SLICE/2)
END
thindbh         2019    PARMS(DBHMIN2,DBHMAX2,CUTEFF,ALL,0,0)
```

Figure 2.3: FVS keyword file code (R1).

DBHMIN and DBHMAX are lower and upper bound values respectively for DBH values that could realistically occur in a southeastern forest. DBHMIN2 and DBHMAX2 define the actual DBH range to be cut in simulation. This range is a calculated percentage of the maximum possible range and is determined by the values of STP and SL. The last line in the FVS code shows how the information encoded in a chromosome has been interpreted for simulation. The simulator will perform a thinning treatment (thindbh) on the stand, cutting a percentage defined by CUTEFF on trees with a DBH between DBHMIN2 and DBHMAX2.

The second representation (R2) also specifies treatments with three floating point numbers <CUTTEFF, VAR1, VAR2> in the range [0, 1]. The allele for cutting efficiency serves the same purpose as in the first representation, but the other two numbers specify the min and max DBH as illustrated in Figure 2.4.

```
COMPUTE         2019
CUTEFF = 0.5
VAR1 = 0.6
VAR2 = 0.25
DBHMAX = 56
STARTDBH = (VAR1*DBHMAX)
DELTA = DBHMAX - STARTDBH
ENDDBH = (VAR2*DELTA)+STARTDBH
END
thindbh         2019    PARMS(STARTDBH,ENDDBH,CUTEFF,ALL,0,0)
```

Figure 2.4: FVS keyword file code (R2).

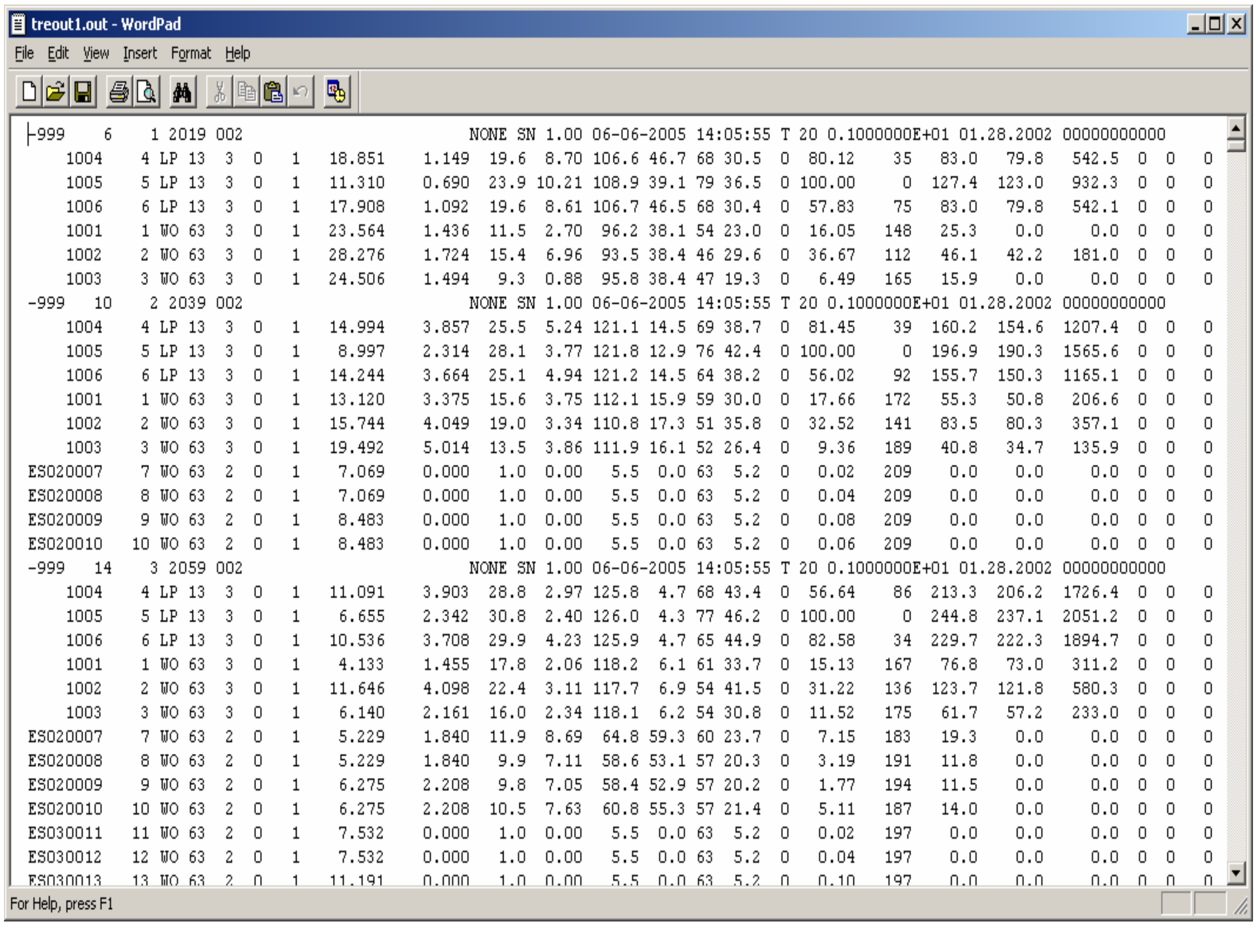
DBHMAX is a species dependent value indicating the largest diameter trees of that species grow to. Like MAXDBH in the first representation, this serves as an upper bound on the range of DBH values that may be included in a treatment. The computed range from STARTDBH to ENDDBH defines the actual range of DBHs to be thinned.

Importantly, in both of these representations each allele corresponds directly and continuously to a particular feature of the phenotype. Every allele codes for a discrete, observable feature in the plan, and small changes in genotype will result in small changes to the phenotype. In the first representation for example, changing the value of STP shifts the range of trees to be cut towards smaller or larger trees, while SL determines how wide or narrow the range will be. CUTEFF tells the program what percentage of trees in the range to cut. These three simple variables can be altered and recombined to represent a great variety of thinning treatments.

2.2 FITNESS FUNCTION

One of the most difficult steps in the design of a GA is selecting an appropriate fitness function. The role of the fitness function is to assign numeric values to solutions in the population indicating how good, or “fit”, those solutions are so that selection operators can be applied. Gong (1992) gives a good account of the difficulties involved in eliciting a suitable fitness function from domain experts. Forest management experts may be unwilling or unable to quantify the criteria by which they judge the desirability of various forest conditions. Fortunately work has already been done at the University of Georgia to develop rules for goal analysis in the context of the NED project. These rules, while not definitive, have a sound theoretical basis in expert opinion (Rauscher et al., 2000).

Data used as input for the fitness function is extracted from the FVS tree list output file (*.out). The tree list contains tree records generated by the simulator for each treatment cycle. These records include attributes such as DBH values and trees per acre that are needed to evaluate fitness. Figure 2.5 shows a sample output file. The rows beginning with “-999” are machine headers and contain information such as the year of the treatment (the fourth field) and cycle length (field 11). The rows below the machine header are the tree records. The GA uses the data from four of the columns: The third column contains a code indicating the species of tree for an observation; data in the fifth column are tree class codes indicating timber quality; the eighth column lists trees per acre; and the 11th column contains the DBH in inches for each observation (Dixon 2002).



```
-999     6     1 2019  002                          NONE SN 1.00 06-06-2005 14:05:55 T 20 0.1000000E+01 01.28.2002 00000000000
     1004     4 LP 13   3   0   1    18.851     1.149  19.6  8.70 106.6 46.7 68 30.5  0  80.12     35   83.0   79.8   542.5  0  0   0
     1005     5 LP 13   3   0   1    11.310     0.690  23.9 10.21 108.9 39.1 79 36.5  0 100.00      0  127.4  123.0   932.3  0  0   0
     1006     6 LP 13   3   0   1    17.908     1.092  19.6  8.61 106.7 46.5 68 30.4  0  57.83     75   83.0   79.8   542.1  0  0   0
     1001     1 WO 63   3   0   1    23.564     1.436  11.5  2.70  96.2 38.1 54 23.0  0  16.05    148   25.3    0.0     0.0  0  0   0
     1002     2 WO 63   3   0   1    28.276     1.724  15.4  6.96  93.5 38.4 46 29.6  0  36.67    112   46.1   42.2   181.0  0  0   0
     1003     3 WO 63   3   0   1    24.506     1.494   9.3  0.88  95.8 38.4 47 19.3  0   6.49    165   15.9    0.0     0.0  0  0   0
-999    10     2 2039  002                          NONE SN 1.00 06-06-2005 14:05:55 T 20 0.1000000E+01 01.28.2002 00000000000
     1004     4 LP 13   3   0   1    14.994     3.857  25.5  5.24 121.1 14.5 69 38.7  0  81.45     39  160.2  154.6  1207.4  0  0   0
     1005     5 LP 13   3   0   1     8.997     2.314  28.1  3.77 121.8 12.9 76 42.4  0 100.00      0  196.9  190.3  1565.6  0  0   0
     1006     6 LP 13   3   0   1    14.244     3.664  25.1  4.94 121.2 14.5 64 38.2  0  56.02     92  155.7  150.3  1165.1  0  0   0
     1001     1 WO 63   3   0   1    13.120     3.375  15.6  3.75 112.1 15.9 59 30.0  0  17.66    172   55.3   50.8   206.6  0  0   0
     1002     2 WO 63   3   0   1    15.744     4.049  19.0  3.34 110.8 17.3 51 35.8  0  32.52    141   83.5   80.3   357.1  0  0   0
     1003     3 WO 63   3   0   1    19.492     5.014  13.5  3.86 111.9 16.1 52 26.4  0   9.36    189   40.8   34.7   135.9  0  0   0
ES020007      7 WO 63   2   0   1     7.069     0.000   1.0  0.00   5.5  0.0 63  5.2  0   0.02    209    0.0    0.0     0.0  0  0   0
ES020008      8 WO 63   2   0   1     7.069     0.000   1.0  0.00   5.5  0.0 63  5.2  0   0.04    209    0.0    0.0     0.0  0  0   0
ES020009      9 WO 63   2   0   1     8.483     0.000   1.0  0.00   5.5  0.0 63  5.2  0   0.08    209    0.0    0.0     0.0  0  0   0
ES020010     10 WO 63   2   0   1     8.483     0.000   1.0  0.00   5.5  0.0 63  5.2  0   0.06    209    0.0    0.0     0.0  0  0   0
-999    14     3 2059  002                          NONE SN 1.00 06-06-2005 14:05:55 T 20 0.1000000E+01 01.28.2002 00000000000
     1004     4 LP 13   3   0   1    11.091     3.903  28.8  2.97 125.8  4.7 68 43.4  0  56.64     86  213.3  206.2  1726.4  0  0   0
     1005     5 LP 13   3   0   1     6.655     2.342  30.8  2.40 126.0  4.3 77 46.2  0 100.00      0  244.8  237.1  2051.2  0  0   0
     1006     6 LP 13   3   0   1    10.536     3.708  29.9  4.23 125.9  4.7 65 44.9  0  82.58     34  229.7  222.3  1894.7  0  0   0
     1001     1 WO 63   3   0   1     4.133     1.455  17.8  2.06 118.2  6.1 61 33.7  0  15.13    167   76.8   73.0   311.2  0  0   0
     1002     2 WO 63   3   0   1    11.646     4.098  22.4  3.11 117.7  6.9 54 41.5  0  31.22    136  123.7  121.8   580.3  0  0   0
     1003     3 WO 63   3   0   1     6.140     2.161  16.0  2.34 118.1  6.2 54 30.8  0  11.52    175   61.7   57.2   233.0  0  0   0
ES020007      7 WO 63   2   0   1     5.229     1.840  11.9  8.69  64.8 59.3 60 23.7  0   7.15    183   19.3    0.0     0.0  0  0   0
ES020008      8 WO 63   2   0   1     5.229     1.840   9.9  7.11  58.6 53.1 57 20.3  0   3.19    191   11.8    0.0     0.0  0  0   0
ES020009      9 WO 63   2   0   1     6.275     2.208   9.8  7.05  58.4 52.9 57 20.2  0   1.77    194   11.5    0.0     0.0  0  0   0
ES020010     10 WO 63   2   0   1     6.275     2.208  10.5  7.63  60.8 55.3 57 21.4  0   5.11    187   14.0    0.0     0.0  0  0   0
ES030011     11 WO 63   2   0   1     7.532     0.000   1.0  0.00   5.5  0.0 63  5.2  0   0.02    197    0.0    0.0     0.0  0  0   0
ES030012     12 WO 63   2   0   1     7.532     0.000   1.0  0.00   5.5  0.0 63  5.2  0   0.04    197    0.0    0.0     0.0  0  0   0
ES030013     13 WO 63   2   0   1    11.191     0.000   1.0  0.00   5.5  0.0 63  5.2  0   0.10    197    0.0    0.0     0.0  0  0   0
```

Figure 2.5: Example of tree list output file (*.out).

Before a goal analysis can be performed on a stand it must be assigned a forest and prescription type. The expert knowledge needed to perform these classifications is borrowed from NED-2. Forest types are labels such as "pine" or "mixed pine-hardwoods" that reflect the species composition of the stand. The forest type determines the prescription type for the stand, which in turn determines which rules will be used during goal analysis.

The NED-2 knowledge base also contains rules that define goals in terms of desired future conditions (DFCs). Simply put, DFCs are conditions that are necessary and sufficient for satisfying a management goal. NED-2 uses DFCs to evaluate the success or failure of user-defined treatment plans in achieving a set of objectives. The point is to give the user who has designed and simulated a plan some feedback on how likely his or her plan is to succeed at meeting the management objectives.

Silvicultural goals have different DFCs depending on the prescription type of the forest. For example, if a user selects the timber goal “Focus on cubic foot production”, then a stand classified as an aspen-birch forest type meets the goal if the total basal area and acceptable growing stock for timber exceed certain thresholds. The thresholds may have different values for the various prescription types depending on the characteristics of the associated forest types. Nute et al. (2000) and Rauscher, Lloyd, Loftis, and Twery (2000) provide an extensive discussion of the hierarchy of DFCs and goals in NED-2.

Viewing goal analysis as a constraint satisfaction problem in which certain DFCs either are or are not met has notable advantages over a system that searches for an optimum solution. First, assuming that certain crucial thresholds are satisfied, differences between candidate solutions may be inconsequential. Forest managers in the real world may not care about finding optimal solutions, just good solutions. Second, forest ecosystems are complicated entities and assuming certain basic criteria are met it may not always be possible to confidently rate one solution more highly than another. The third reason is a bit more complex and relates to a concept in multi-objective optimization called *Pareto optimality*. The object of single objective optimization problems is to find a particular point in the search space that gives the best value for that objective. By contrast, during *n*-objective optimization there may be no one point that is superior to all other solutions, but rather an *n*-1 dimensional surface called the *Pareto optimal front* where no point on the surface dominates any other (Smith, 1994). For example, if the GA has two goals, “periodic income” and “enhance big tree appearance”, there might not be any one best solution. One good solution might satisfy the first goal with a score of 4.0 and the second with a score of 5.0, while another good solution satisfies the first with a score of 5.0 and second with a score of 4.0. In this case the Pareto optimal front would consist of the set of solutions with scores of 9.0, and could be plotted as a line on a graph. Under the DFC system of goal analysis goals may be satisfied with varying degrees of confidence. This is very convenient because it allows goals that may deal with different units or quantities behind the scenes to be dealt with in like terms; namely, how confident the system is that the goals were satisfied during each year of the plan.

The fourth and perhaps most important advantage to using DFCs is that it allows the reasons for a goal's success or failure to be easily explained to the user, giving managers a clue as to what went wrong and what, if anything, can be done to correct the shortcoming (Rauscher et al., 2000).

Five DFC-based goals from the NED-2 knowledge base have been adapted into a fitness function for the GA. Four of these goals involve timber production, while the fifth relates to the visual quality of the forest and was selected to counter-balance the purely economic focus of the other goals. Treatment prescription becomes a more complex and interesting problem when economic needs must be balanced by esthetic or ecological goals. The goals are: focus on cubic foot production, focus on board foot production, periodic income, focus on net present value, and enhance big tree appearance (see Appendix A for the content of these rules). Any combination of these goals may be selected for optimization. To determine the fitness of a treatment plan, the GA simulates it with FVS and checks whether the output satisfies the DFCs for the selected goals. The NED-2 help files show how to calculate basal area, relative density, and other variables referred to in the DFCs from tree observation data in FVS *.out files.

One way to determine the fitness of a treatment schedule upon goal analysis is to simply sum the number of goals that are satisfied over the course of the simulation. With this method, a goal is either satisfied for a cycle or it is not. So if two goals are selected for a simulation with 10 cycles of any interval length, the best possible fitness would be 20. One version of the GA has been developed and tested using this simple design.

Unfortunately the target values defined by DFCs are not incontrovertible (Rauscher et al., 2000) and values obtained in simulation are not guaranteed to match reality. Another concern is that using rigid threshold values for goal satisfaction could degrade the performance of the GA. Ideally it should be possible for the GA to make incremental progress towards satisfying goals with a series of small improvements to the composition of the gene pool. If a DFC for a goal states that the stand should have a basal area of 60 square feet per acre and the actual value is 59.9, the GA should recognize that while current conditions fall short of the DFC, this is an improvement over a previous plan that resulted in a basal area of just 10 square feet per acre.

To help mitigate these concerns, confidence factors (CF) indicating the degree to which a goal or DFC is satisfied have been incorporated into goal analysis. Confidence factors capture the idea that goal satisfaction can operate across a continuum. For example, if a goal is satisfied well within the ranges defined by its DFCs it is assigned a higher value than a goal that is barely satisfied. CFs are assigned by partitioning observed values into categories relative to thresholds ± 10% of the DFC's target value. A DFC may be completely satisfied (CF = 1.0), marginally satisfied (CF = 0.6), nearly satisfied (CF = 0.4), or it may fail completely (CF = 0.0) (Routh, 2004).

Most goals are defined by three or more DFCs, each with its own CF. The values associated with each DFC are combined to produce a number indicating the degree to which the whole goal is satisfied.

For two conditions P1 and P2:

CF(P1 and P2) = min((CF(P1), CF(P2))

CF(P1 or P2) = max((CF(P1), CF(P1))

At the time of writing the premise for every goal in the GA is a conjunction of DFCs, so the degree to which any goal is satisfied is always the same as the CF of the least satisfied DFC. The fitness of a stand is determined by adding the CFs of all the goals that were at least marginally satisfied over the course of the plan. This involves checking the values assigned to goals during each treatment interval, or cycle, of the plan, then summing those values.

Checking the status of management goals during each cycle, as opposed to evaluating the goal status at the end of a plan, enables the GA to show a preference for plans that satisfy management goals early and often. If two plans satisfy the same goal set but one satisfies those goals through more cycles than the other, the latter plan will be assigned a higher fitness.

The method described above is the default procedure for calculating the fitness of plans. Two alternative methods of assigning fitness were also explored. A vector evaluated GA, or VEGA, is a very simple modification of a simple GA for multi-objective optimization (Schaffer, 1985). In a VEGA, the population is randomly divided into *N* equal subpopulations every generation, where *N* is the number of goals. The fitness of each subpopulation is calculated based on a different objective function. This method helps prevent dominance by individual champions of a particular goal. When all goals are considered simultaneously it is possible for a solution to dominate the population by virtue of its success at satisfying a particular goal very well, even if it does not do well by the other objectives. In a VEGA this is less likely to occur because the function the solution is measured against is randomly determined by its assignment to one of the subpopulations. An individual that easily satisfies one goal at the expense of others is unlikely to do well under a VEGA system. Another advantage to checking solutions against one objective function at a time is that there is no problem with differences in the ranges of objective functions. If one goal is more easily satisfied than another, performance on that objective might drown out the fitness contribution from another goal if all objectives were considered simultaneously. With VEGA solutions that perform well on many goals should have a distinct advantage, while those that are not particularly fit overall but satisfy a particularly difficult goal may persist in the population as management alternatives.

Another GA modification for multi-objective optimization, first suggested by Bentley and Wakefield (1997), uses a ranking scheme to encourage a variety of solutions in the GA population. The weighted average ranking (WAR) method ranks population members separately according to their performance on each objective function. The fitness of a solution is assigned by summing its rankings (in this system, a higher number is a better rank). Like the VEGA, the WAR scheme is intended to prevent dominance of the population by individual champions.

2.3 GENETIC OPERATORS

One of the fundamental problems encountered when trying to optimize the design of a GA is balancing the needs for population diversity and strong selection pressure. Ideally the GA should find optimal or near-optimal solutions, but the final population should be diverse enough that the program can offer alternative recommendations. The difficulty is that the strength of selection pressure is often inversely related to the amount of diversity in the population. Applying mutation and crossover operations directly to real-parameter values also presents a challenge. The goal of genetic operators is to perturb the allele pool in incremental and meaningful steps, discovering and propagating meaningful schemata. This section presents a discussion of these issues and explains the reasoning behind the choice of genetic operators.

2.3.1 MUTATION

Mutations are chance alterations in the genetic composition of offspring as they are created from parents in the mating pool. This serves an important and complementary role to crossover. While the latter recombines genetic information from parent solutions into new configurations, mutation represents the random error in copying that sometimes occurs in nature. Sometimes these errors convey significant advantage to an individual and become prominent schemata in the population. Mutation may play an important role in the success of the GA because it samples allele values that do not occur anywhere in the initial population and would thus not be explored by a conventional crossover. Although the importance of mutation is less prominent in a GA with blending crossovers (discussed in the next section), guided mutation can still be a valuable tool for tweaking values on an allele-by-allele basis.

In the current study mutation is usually incremental. This is necessary because small changes in a treatment plan genotype, in the allele for cutting efficiency for example, can have a large impact on the fitness of a treatment plan. Since all alleles are percentages between zero and one, an incrementally mutated allele adds or subtracts 0.05 from its previous value.

It appears probable that, on average, not all legal allele values are equally likely to be parts of effective treatment definitions. Clearcutting and the grow condition (where nothing is done because the cutting efficiency is set to 0) in particular seem likely to be common treatments when management goals are timber-related. Plans with instructions to simply let a stand grow or to clearcut also have the advantage of being parsimonious. If two plans satisfy the same goal set, but one involves several precise cutting treatments over narrow DBH ranges while the other recommends letting the stand grow for several cycles and then clearcutting, the latter plan will be the more attractive option of the two because it will be easier and more economical to execute. For these reasons it may be desirable to occasionally mutate treatments into grow or clearcutting treatments instead of the usual incremental mutation.

Treatments for a cycle are represented over three alleles. Therefore the mutation operator must look at three alleles at a time. In the case of a grow mutation the first allele, which codes for cutting efficiency, is set to 0. The second and third alleles are left alone. Mutation into a clearcut changes the alleles differently depending on which representation is used. In R1 CUTEFF, STP and SL change to 1.0, 0.5, and 1.0 respectively. For R2 the alleles coding for CUTEFF, VAR1 and VAR2 become 1.0, 0, and 1.0. 20% of all mutations result in a grow treatment, 20% create clearcuts, and the remaining 60% of the time each allele is independently evaluated with possible tweaks of plus or minus 0.05, as described before. The 20/20/60 breakdown is not theoretically grounded but produced reasonable results in testing. Figure 2.6 illustrates the three mutation types. All test runs were performed with a fairly standard baseline mutation rate of 0.05.

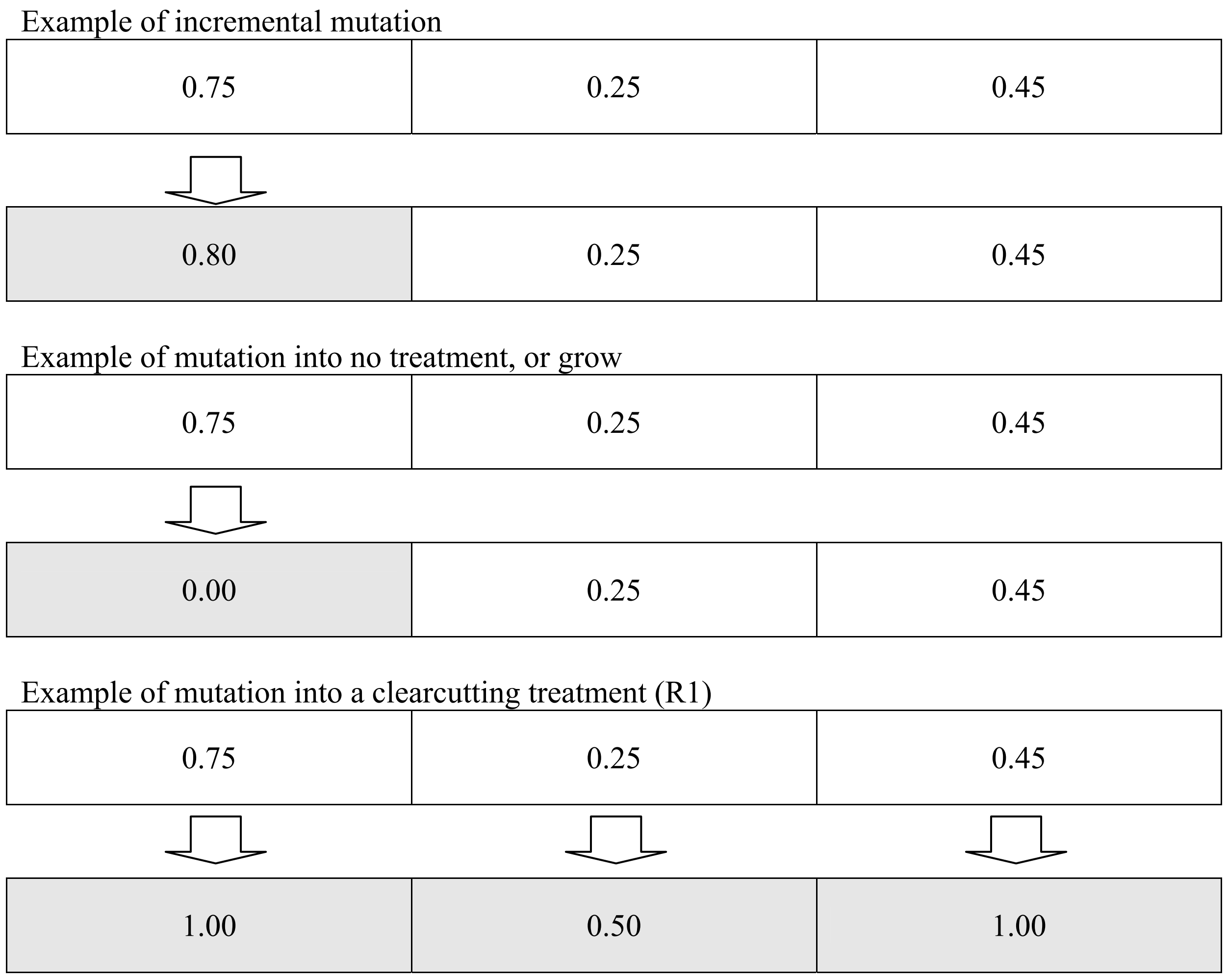

Example of incremental mutation
0.75
0.25
0.45
0.80
0.25
0.45
Example of mutation into no treatment, or grow
0.75
0.25
0.45
0.00
0.25
0.45
Example of mutation into a clearcutting treatment (R1)
0.75
0.25
0.45
1.00
0.50
1.00

Figure 2.6: Illustration of mutation types.

2.3.2 CROSSOVER

Crossover is an operation that separates and recombines parent solutions to create two new solutions with a mix of characteristics from each parent. Ideally if two solutions selected for mating each have desirable characteristics, the offspring created by crossover will inherit the best features from both. The solutions created by lucky crossovers may be significantly more fit than either of the solutions from which they are constructed.

Early versions of the GA were implemented with two-point and uniform crossover operators. In two-point crossover, two allele positions on a parent chromosome are selected as crossover points. The section of chromosome between the crossover points is swapped with the corresponding section from another individual, resulting in two new chromosomes, or offspring. For example, a two-point crossover on parents P_1 and P_2 where $P_1 = [1, 2, 3, 4, 5]$ and $P_2 = [6, 7, 8, 9, 10]$ at the second and fourth positions would result in offspring O_1 and O_2, where $O_1 = [1, 2, 8, 9, 5]$ and $O_2 = [6, 7, 3, 4, 10]$.

In uniform crossover a crossover mask is generated that defines which positions will be swapped between chromosomes. The mask [1, 5] applied to the same parents from the previous example would result in offspring O_1 and O_2, where $O_1 = [6, 2, 3, 4, 10]$ and $O_2 = [1, 7, 8, 9, 5]$.

Every allele in a binary-coded GA has one of two values: one or zero. Mutation can flip an allele value from one bit to another, and the crossover operator searches different schema orderings. Unfortunately traditional crossover methods like two-point and uniform may be less effective when dealing with real-parameter GAs. The reason for this can be understood by considering the dimensions of the search space. Because the value of each allele can vary continuously between zero and one, it is vital not only to explore different possible orderings of existing alleles, but also to try new allele values. Conventional crossover methods that simply reorder the alleles from parent chromosomes place too much of the burden of the search on the mutation operator. No matter how the alleles in a real-parameter GA are shuffled position-wise, if different values are not sampled large portions of the search space will be neglected.

For these reasons it may be necessary to explore alternative crossover options. An effective operator needs to explore the ranges of continuous values in string positions, creating offspring equidistant in the search space from the parent chromosomes. Technically this is better described as a blending operation instead of crossover, although for the sake of simplicity the term crossover will continue to be used.

One of the simplest real-parameter crossover implementations is known as linear crossover (Wright, 1991). This operator creates three candidate offspring, $0.5(P_1 + P_2)$, $(1.5P_1 - 0.5P_2)$ and $(-0.5P_1 + 1.5P_2)$ from parent chromosomes P_1 and P_2. For example, a linear crossover on parents P_1 and P_2 where $P_1 = [0.5, 0.5, 0.5]$ and $P_2 = [1.0, 1.0, 1.0]$ results in offspring O_1, O_2, and O_3 where $O_1 = [0.75, 0.75, 0.75]$, $O_2 = [0.25, 0.25, 0.25]$, and $O_3 = [1.25, 1.25, 1.25]$. Typically the better two of the three are passed to the next generation. In this case O_3 is an illegal crossover result because all allele values in the GA are required to be in the range [0, 1], so O_1 and O_2 would be selected.

Eshelman and Schaffer (1993) suggested a different method, called blend crossover (BLX-α) for real-parameter GAs. For two parent chromosomes P_1 and P_2, and assuming $P_1 < P_2$, BLX-α creates offspring in the range $[P_1 - \alpha(P_2 - P_1), P_2 + \alpha(P_2 - P_1)]$. The value of α determines the width of the range of possible offspring values. Thus if $\alpha = 0$, the crossover will result in a random solution in the range $[P_1, P_2]$. The investigators reported that a BLX with $\alpha = 0.5$ performed the best on several test problems, so this is the value used in our GA.

One interesting characteristic of BLX-α is that the range of values in offspring depends on the difference between parent solutions. When the parents are far apart in the search space the difference between parents and offspring is also large, but as the population converges, new offspring are more similar to their parents. This is a desirable characteristic in a genetic operator; a larger space of solutions will be searched early on, and as the population converges the search will become appropriately focused.

The third and final real-parameter crossover operator explored in this study is simulated binary crossover (SBX), which was developed by Deb and his students (Deb & Agrawal, 1995; Deb & Kumar, 1995). SBX was designed to approximate the effect of single-point crossover on binary strings for non-binary representations. Essentially SBX creates a probability density function such that offspring values closer to the values of the parents are more likely to occur than values far from the parents. Offspring are symmetric around parent solutions in the search space to avoid bias towards either parent. The distribution index η_c affects the steepness of the probability curve, with high values steepening the curve and giving a higher probability to near-parent solutions. Like BLX-α, the difference between offspring is proportional to the difference between the parent solutions. However, SBX has the additional property that near-parent solutions are biased over distant solutions. Put another way, BLX-α narrows the search space as parent solutions become more similar, but within the range of legal values all offspring are equally probable. SBX does not restrict the range of possible values, but rather makes some values more likely than others.

The baseline crossover rate for our GA is 0.6. This number is only meaningful during early generations however, because of a design decision to enforce genotypic uniqueness in the population. This is discussed further in the next section.

2.3.3 SELECTION

There are several methods for selecting individuals for the mating pool once fitness values have been assigned. The main objective is keeping selection pressure high, meaning favoring solutions with high fitness values, while simultaneously maintaining a healthy amount of diversity in the population. When selection pressure is high individuals with small differences in phenotype may be evaluated very differently. One way of achieving this is by taking the exponent of raw fitness values, so that even if an individual is only slightly better than its neighbors, it will be much more likely to be selected for the mating pool.

The problem with high selection pressure is its inverse relationship with population diversity. Diversity in the population is desirable because it allows the search space to be more fully explored, and because the population at the last generation should ideally include several alternative solutions representing significant management alternatives. If selection pressure is too severe, especially in the early generations, the population can become dominated by a small number of super-individuals, and the subsequent lack of richness in the allele pool may stymie further improvements. This is the problem of getting stuck in a local optimum; sometimes it is necessary to temporarily move through a "valley" in the search space to reach the highest peak. On the other hand, if selection pressures are too low, diversity may be preserved but population convergence towards good solutions may be very slow. The difficulty is exacerbated by the fact that GAs usually do well with relatively low selection pressure in early generations, but populations become more competitive in later generations, resulting in a need for higher selection pressure. This has come to be known as the scaling problem.

Whitley (1989) advocates a rank-based selection scheme as the best way to fine tune this balance. In his scheme, the likelihood that an individual will be selected for the mating pool is determined by its fitness ranking compared to other members of the population. The advantage of using fitness rankings rather than actual fitness values is that selection pressure remains level throughout the run of the GA. The most fit individual has the same advantage over the least fit individual whether the difference in fitness values is 1 or 100. This has the effect of helping to prevent premature convergence in early generations while continuing to reward even small improvements in later generations when the population members are closer in fitness. For these reasons rank-based selection has been selected for use in the current study.

2.3.4 ENFORCED GENOTYPIC UNIQUENESS AND ELITISM

Another potentially important design decision is whether to enforce genotypic uniqueness in the population, and if so how to go about it. Intervening in the normal run of the GA to ensure that no duplicate individuals exist seems to run counter to the general GA philosophy of letting artificial evolution run its course. Thus some explanation is in order.

One potential benefit of enforcing uniqueness is that it works as a foil to the dominance of super-individuals in early generations. Although this prevents the GA from early convergence on a single point in the search space, a particular class of individual may still dominate if every population member is a closely related variation of one solution. Another problem with this approach is that it makes the relative contributions of the mutation and crossover operators very difficult to interpret, because crossovers resulting in duplicates of existing members of the population are repeated until novel offspring are created. In spite of these drawbacks, enforced genotypic uniqueness may be useful in creating a range of non-dominated alternative solutions with similar fitness values, especially when combined with a sophisticated crossover operator. Whitley (1989) also notes that selection procedures are most fair in populations with no duplicate individuals. An individual with multiple copies in the population is more likely to be selected whatever its fitness value may be. When each solution is unique, selection for the mating pool is determined entirely by individuals' relative fitness values.

The selection operation works as follows. During the formation of a new generation individuals from the current population are selected in pairs by the rank-based selection method to create offspring. If the set crossover rate (0.6) is greater than a randomly selected number between zero and one, a crossover operation is applied to the parents which results in two offspring for the new population. Otherwise the parents themselves are passed into the new population. The exception to this procedure occurs when a duplicate solution is passed into the new generation (assuming the option to enforce genotypic uniqueness is enabled). In this situation the program backtracks and repeats the crossover operation until a unique solution is created.

Elitism has also been implemented in the GA. Elitism is a mechanism that guarantees the most fit individual in a population a place in the next generation. When a new population is spawned, the GA automatically selects the most fit N percent of individuals from the current population first. The rest of the new population is filled out by the normal selection, crossover, and mutation procedure.

CHAPTER 3
EXPERIMENTS

Several experimental runs of the GA were conducted to decide between alternative genetic operators and selection methods. The two alternative representations discussed in section 2.1 were also compared. Having decided on the parameter settings, the performance of the GA was tested against a hill-climbing/GA hybrid, simulated annealing, and steady state GA across two conditions, with and without CFs, for a total of eight test conditions. In the final experiment a plan generated by the GA was compared to a plan recommended by a human expert.

The management units mentioned in this section (Deer Hill, Liberty 1, and Bent Creek) are data sets containing forest inventory information that are used for testing NED-2. With two exceptions the experiments were simulated using data from 10 stands in the Deer Hill management unit located in Williamsburg County, South Carolina. Figure 3.1 shows the default settings used in these experiments. The settings were held constant unless otherwise noted.

The experiment described in 3.3 was simulated on the Liberty 1 dataset, from Liberty Reservoir in Baltimore County, Maryland, which consists of five stands. This experiment compared methods intended for evaluating fitness in systems with multiple goals, so every available goal was selected. Liberty 1 was used instead of Deer Hill because the former offers greater opportunity for the more difficult goals to be satisfied.

The experiment described in 3.6 (comparison to a human expert) was conducted using the Bent Creek dataset at the request of the expert, and the evaluation procedure was significantly more complicated than for other experiments. This is explained in more detail in the sections ahead.

Statistical analysis of results was performed using a two-factor analysis of variance (ANOVA) with replication with alpha = 0.05. This tests the hypothesis that data from two or more groups are drawn from populations with the same mean (the null hypothesis), where there are multiple samples for each group of data. A p-value is the probability that a sample would have been selected assuming that the null hypothesis is true. The value of alpha sets the confidence level for the test. A p-value less than 0.05 for example indicates a significant difference between groups with 95% confidence.

Every experimental condition was tested 10 times for each of the stands in the management area. For example, in the experiments run on the 10 stands from Deer Hill each of the 10 samples consisted of 10 rows of data, one row per stand. Table 3.1 shows an example of how the data was formatted for ANOVA. The two-factor test was selected to account for the possibility that there might be an interaction between the stand number and the variables of interest due to differences in stand compositions. Although 10 samples per condition is not ideal for measuring statistical differences between groups, time constraints limited the number of tests that could be performed. Fortunately, during testing it was observed that stand composition never had a significant interactive effect with the experimental variables under consideration. Generally speaking, what worked well for one stand worked well for all stands.

```
Database: DeerHill
Stands Processed: [1,2,3,4,5,6,7,8,9,10]
Population Size: 50
Max Generations: 40
FVS Cycle Length: 20
FVS Number of Cycles: 10
Representation: R1
Selection Type: rank
Mutation Rate: 0.05
Crossover Rate: 0.6
Crossover Type: uniform
Elitism (Percent): 2
Genotypic Uniqueness Enforced: yes
Fitness Function: DFCs with confidence factors
Goals: Periodic income, enhance big tree appearance
```

Figure 3.1: Default settings for generational GA; experiments 3.1-3.4.

Table 3.1: Example of data formatted for two-factor ANOVA with replication. Each row is data from one stand. Each cell contains the best fitness value found for a stand on a particular trial. Actual experiments consisted of 10 trials.

	GGA	*SA*	*HC*	*SSGA*
Trial 1	14	8	10	12
	12	10	9	13
	8	11	8	9
	15	14	12	15
	14	14	10	13
	14	15	13	13
	14	15	11	14
	13	13	11	11
	11	11	13	12
	12	12	7	13
Trial 2	13	12	10	13
	11	14	11	12
	8	9	8	12
	13	12	12	13
	15	14	12	13
	16	15	13	13
	14	12	11	14
	15	13	10	13
	12	11	7	11
	14	13	10	10

3.1 COMPARISON OF REPRESENTATIONS

The two representations described in section 2.1 were compared on the GA with no CFs (this experiment was conducted before their implementation). All other parameters were set to default values. To review, treatments in the first representation (R1) are triples <CUTEFF, STP, SL>, where CUTEFF defines the percentage of trees in the range to be cut, STP is a percentage representing the midpoint of the range of DBHs to be cut, and SL is a percentage used to determine the width of the cutting range. The second representation (R2) also represents treatments with three floating-point numbers between zero and one, <CUTEFF, DBHMIN, DBHMAX>, but the treatment range is computed from the second and third arguments differently than in R1.

The results from R1 and R2 were not significantly different ($p = 0.71$; $p > 0.05$). Table 3.2 shows results by stand number. The cell values are the means of the best fitness values found during each of the 10 runs of the GA. For consistency, R1 was used as the default setting for the remainder of the experiments.

Table 3.2: Comparison of representations.

Stand	*R1*	*R2*
1	14.1	14.1
2	11.6	12.8
3	9.8	7.5
4	14	13.8
5	14.4	12.1
6	15.6	15.8
7	14	13.6
8	13.9	16
9	12.1	13.2
10	13.3	15.2
All Stands	**132.8**	**134.1**

3.2 COMPARISON OF CROSSOVERS

The next set of experiments tested the relative effectiveness of different crossover operations. The primary purpose of these experiments was to compare classical crossover techniques to BLX-α and SBX, which were specifically developed for use with real-parameter GAs.

Another point of interest was whether the blending operations of BLX-α and SBX would be more effective when applied on an allele-by-allele basis or across entire chromosomes simultaneously. This issue arose for two reasons. First, both operators use random numbers in calculations that create offspring allele values from parent alleles. One question addressed by this experiment is whether better results are achieved when a different random number is used for each allele position, compared to using the same number across an entire chromosome. The other reason for creating offspring one allele at a time is that the operations of BLX-α and SBX sometimes result in illegal alleles, that is, alleles with values greater than one or less than zero. If every offspring allele position is calculated independently, illegal alleles can be re-blended until a legal value is generated. Illegal alleles are much more problematic when entire chromosomes are blended at once, because it means the offspring must be either entirely recalculated (Option 1) or thrown out, in which case a clone of a parent takes the aborted offspring's place in the next generation (Option 2).

Two versions of BLX-α were tested, one using the allele-by-allele calculation and the other the "whole chromosome" method. The allele-by-allele calculation won by default because Option 1 was impractically slow and Option 2 resulted in a significantly lower crossover rate than the specified 0.6 because it had to throw out so many illegal offspring.

Next the effectiveness of five crossovers: two-point, uniform, linear, BLX-α, and SBX; were compared while other variable parameters were held constant. Although the choice of crossover was significant with $p < 0.001$, the results in Table 3.3 and Figure 3.2 show that the expected advantage for real-parameter crossovers did not materialize (again, results show mean fitness values across all trials). In fact, although the differences were not dramatic, when the best results from all stands were summed for each crossover the two-point and uniform crossover slightly outperformed the real-parameter crossovers. The results from uniform crossover were significantly better ($p = 0.004$; $p < 0.05$) than the results from the best-performing of the real-parameter crossovers, BLX-α. Because two-point and uniform cross can not discover allele values not present in the initial population, the results seem to indicate that either 1) the initial populations already contained the necessary schemata to locate good plans, or 2) those schemata were located by the mutation operator.

Although blending crossovers explore a great diversity of solutions, this comes at the cost of losing the exact allele values the parent solutions comprise. During a linear cross, to take the simplest example, if parents P_1 and P_2 have values at the nth allele position of 1.0 and 0.5, then the offspring may inherit the average of these values (0.75) at position n. Viewing the blending of allele values in terms of the Schema Theorem, it may be that for this particular application the blending operations were too disruptive and ended up destroying important schemata. Uniform cross was selected for the remaining experiments.

Table 3.3: Comparison of crossovers.

Stand	*two-point*	*uniform*	*linear*	*BLX-α*	*SBX*
1	14.42	15.02	13.7	13.18	13.22
2	13.52	13.86	13.18	13.1	13.08
3	11.18	10.88	10.98	11.12	11.14
4	15.38	15.04	14.88	14.98	14.92
5	15.36	15.56	14.48	15.22	14.66
6	15.46	15.22	15.04	15.24	15.18
7	15.24	15.04	14.4	14.72	14.52
8	14.3	14.72	13.66	13.82	13.66
9	12.84	13.54	11.66	11.7	11.74
10	13.12	13.32	12.5	12.7	11.9
All Stands	**140.82**	**142.24**	**134.5**	**135.78**	**134.02**

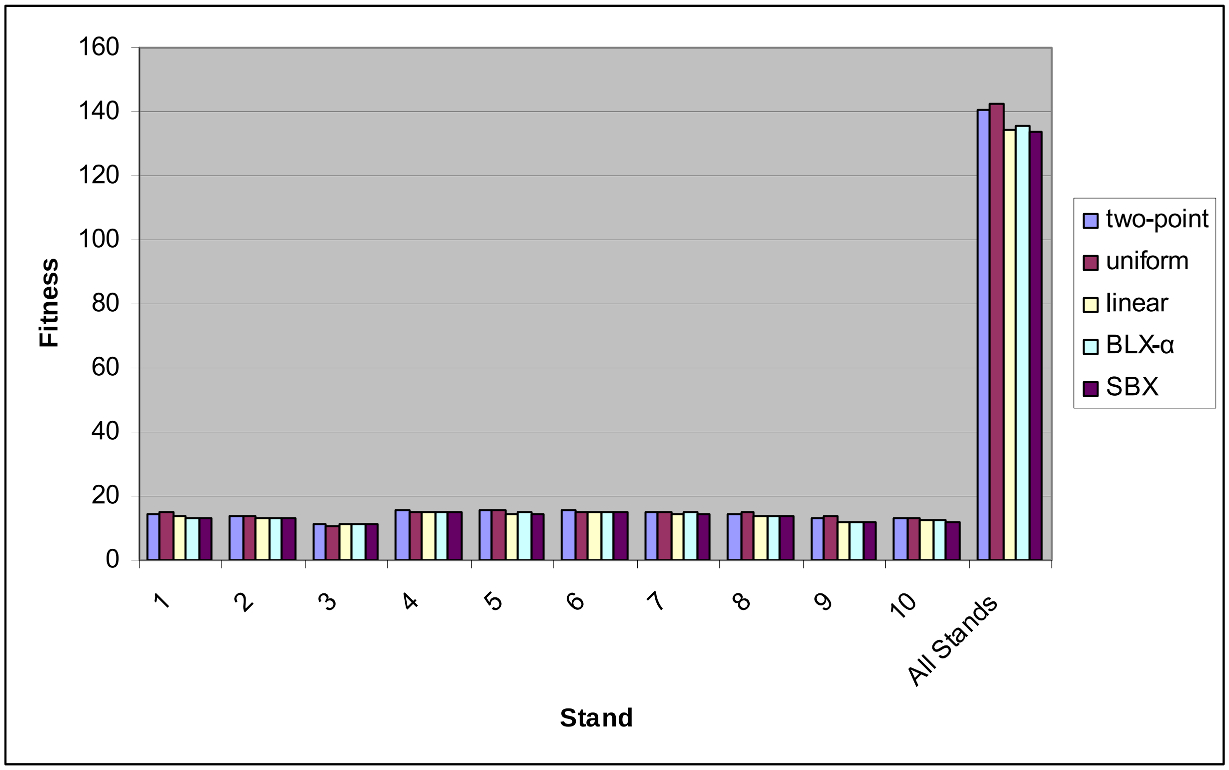


Figure 3.2: Comparison of crossovers.

3.3 COMPARISON OF FITNESS FUNCTIONS

By default fitness is assigned by summing the CFs for each goal from every treatment interval of the plan. This straightforward evaluation procedure was compared to the VEGA and WAR fitness function variants described in section 2.2.

A large part of the appeal of VEGA and WAR is that they provide means for handling goals with incommensurate values. For example, if one goal attempted to maximize timber production in terms of board foot volume, while another goal dealt with inches of leaf litter on the forest floor, it may not be clear how those values should be weighed against one another. In the case of VEGA, the problem disappears because only one goal per individual is evaluated each generation. In the case of WAR, performances on each goal are assigned a rank rather than using absolute values so there is no problem with the different units of measurement. However, because the GA rates individuals according to the confidence with which DFCs are satisfied the problem of incommensurate values does not occur to begin with: The GA's performance on any given goal is assigned a value of 1.0, 0.6, 0.4, or 0.0 only, no matter what units the DFCs deal with.

The remaining incentive for testing VEGA and WAR is that they were designed to prevent dominance of the population by super-individuals in one category. These are solutions that dominate the population in early generations because they do very well on one goal, perhaps at the expense of other goals. If this happens the GA may prematurely converge on a local maximum and subsequently lack the population diversity needed to effectively sample the search space.

Because VEGA and WAR are intended for multi-objective search, all five goals (focus on board foot production, focus on cubic foot production, periodic income, net present value, and enhance big tree appearance) were entered into the fitness function for this experiment. The experiments in this section were conducted on the Liberty data set, which contains five stands. Other parameters such as population size and the treatment interval were left at the default values.

The choice of fitness function across all three conditions was significant. The default fitness function was significantly better than the fitness function using WAR ($p < 0.001$), and WAR was significantly better than VEGA ($p < 0.001$). WAR suffered the additional disadvantage of being the most computationally intensive of the three. This is not surprising because the WAR fitness function must rank each individual's performance relative to the rest of the population on every goal. Table 3.4 shows results by stand number over the 10 trials. Figure 3.3 presents the same information graphically. The regular fitness function was used for the remainder of the experiments.

Table 3.4: Comparison of fitness functions.

Stand	*Normal*	*VEGA*	*WAR*
0	24.16	20.74	21.92
1	25	20.42	20.54
2	24.64	20.24	22.48
3	22.1	18.24	22.18
4	23.92	21.6	23.04
All Stands	**119.88**	**101.24**	**110.16**

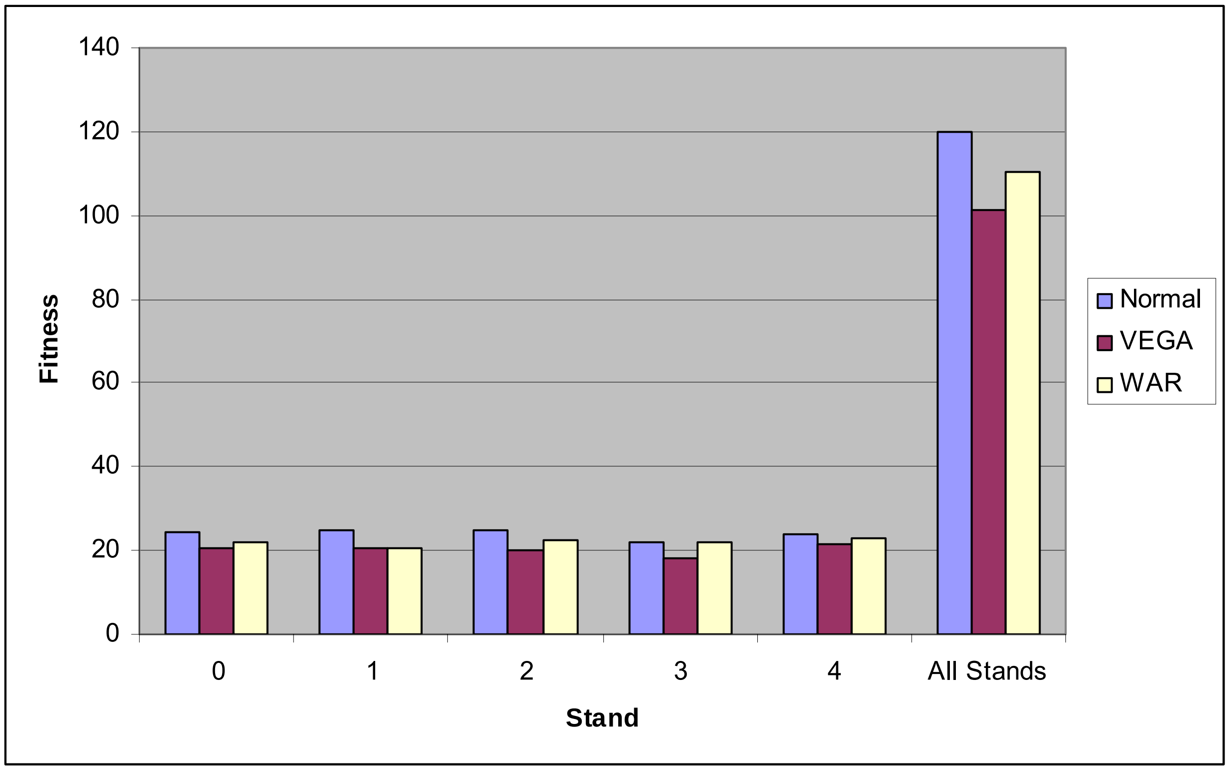


Figure 3.3: Comparison of fitness functions.

3.4 COMPARISONS TO OTHER SEARCH HEURISTICS

In this set of experiments the GA was compared to three other search heuristics: modified hill-climbing, simulated annealing, and the steady-state GA. Each heuristic was tested with and without CFs incorporated into the goal analysis. Without CFs the search space becomes more step-like and discrete, so it would be interesting to see how the searches fare in each condition. The next three subsections introduce the heuristics. The results of the experiments along with a brief discussion are in section 3.4.4.

3.4.1 MODIFIED HILL-CLIMBING

Hill-climbing procedures work by generating a list of possible alterations, or moves, in a search space that can be reached from the current state, and selecting the move that brings the state closest to a goal state. Hill-climbing is a very simple operation that only accepts a new state if it is closer to the goal state than the previous state. The drawback to this approach is that in complex search spaces with many peaks and valleys there is a risk of getting stuck in a local optimum. Hill-climbing is a relatively naive procedure compared to GAs and other evolutionary programming, but with minor modification it was found to be quite effective at finding good treatment schedules.

The routine described here is a hill-climbing/GA hybrid system. First a population of solutions is created, just as in the main GA program, and fitness values are assigned. The most fit individual in the population, I_g is selected as the starting point for the hill-climber. Now begins the recursive part of the program. A new population, analogous to the list of legal moves in a regular hill-climbing algorithm, is created by mutating I_g repeatedly. The most favorable of the mutations is selected as the new state of the system, I_{g+1}. This is repeated for some set number of generations, just as in a GA.

This hybrid system differs from a regular hill-climbing procedure in an important respect. Instead of modifying the current state along a single dimension and moving into a new state if the modification results in a higher fitness, entire plans are subjected to the same mutation operation used by the baseline GA program. Although the majority of mutations result in small phenotypic changes to treatment schedules, multiple allele mutations are a statistical possibility even with a low mutation rate because the operation is performed on an allele-by-allele basis. A significant mutation may actually represent a substantive leap in the topography of the search space, thereby allowing the procedure to escape the local optimums that would entrap a simple hill-climbing routine.

3.4.2 SIMULATED ANNEALING

Simulated annealing (SA) is generalization of the Metropolis Monte Carlo method for evaluating the states of a thermodynamic system. A simple Monte Carlo simulation is a system that samples a search space by randomly choosing different parameters. The results of such sampling provide information about the space.

In a Metropolis Monte Carlo simulation new points in the search space are sampled by making minor changes to the current state of the system. In a thermodynamic system with energy *E* and temperature *T*, if a modification is favorable (negative energy change) the change is accepted, otherwise it may be accepted with a probability given by the Bolzmann factor *exp* – (*dE/T*) (Metropolis et al. as cited in Luke, 2005).

Simulated annealing applies the methodology of a Metropolis Monte Carlo simulation to find the most stable orientation of a system (Kirkpatrick et al. as cited in Luke, 2005). The inspiration for SA is the procedure used in making glass and metals from a melt. At the highest temperatures the molecules in the glass move about relatively easily. As the glass cools the molecules stabilize. If the melt cools slowly the glass adopts a stable orientation, but if cooling occurs too quickly or the initial temperature is too low, defects may occur. By analogy, SA perturbs the current solution to a problem according to a “cooling schedule”. Early in the process large disturbances are permitted for broader search. As the system “cools” changes become less severe and the search becomes focused on an ever smaller region of the search space.

An SA algorithm was implemented and tested against the other methods described in this section. The population corresponds to a collection of individuals around a single point. With the first generation, the best individual is saved. With subsequent generations, the best individual is compared to the stored individual. If the best individual is better than the stored individual, the new individual takes the stored individual's place. If it is inferior, then it is selected with a probability *P* defined by the SA's cooling function. The equation for determining *P* is:

$$P = e^{(-(1-(\Delta f / F))/T)}$$

where *F* is the fitness of the stored individual, *T* is the current temperature, and Δ*f* is the difference in fitness between the new and the stored individuals. The constant *e* is approximated as 2.72.

After P is determined and the stored individual has been updated, a new set of states or moves in the search space are generated and the temperature is decremented. The cooling schedule decreases the temperature by a step size defined as 1/*MAX*, where *MAX* is the number of generations given for the search. Consistent with the GA, the SA search was allowed 40 generations to converge. The initial temperature *T* was set to 1. The hill-climbing algorithm described in the previous subsection is equivalent to an SA search with *T* initialized to 0.

3.4.3 STEADY-STATE GA

The steady-state GA (SSGA) is a variation on standard generational GAs (GGA). Instead of successively spawning new generations from the most fit members of a population, SSGAs continually improve the composition of a single population by replacing the least fit members of the population with the offspring of fit individuals. The SSGA tested here repeats its crossover operation until new offspring have fitness values greater than or equal to the parent chromosomes, providing additional selection pressure.

Chafekar, Xuan, and Rasheed (2004) suggested that SSGAs may be more practical than conventional GAs in situations where determining the fitness of an individual is computationally expensive. In standard GAs a newly formed population will often contain many of the same individuals as the previous population, and the fitness of those individuals will be recomputed multiple times unless special programming steps are taken. SSGAs spend less time computing fitness values than regular GAs because only newly created individuals are evaluated. In the present study a GGA approach was selected at the outset because determining fitness is fairly straightforward, but a simple SSGA was tested for comparison.

The SSGA was run over 450 cycles. This cutoff point is admittedly somewhat arbitrary, but it appeared to be sufficient for convergence to take place. Other parameters, where applicable, were set to the default values.

3.4.4 COMPARISON OF SEARCH HEURISTICS: RESULTS

On the comparison of search heuristics without CFs, the choice of search technique was significant with $p < 0.001$. The best fitness values found for each stand number and search heuristic pairing were averaged over the 10 trials to produce the results shown in Table 3.5. Figure 3.4 presents the same information graphically.

Results on the GGA condition were not significantly different from the SSGA condition ($p = 0.205$; $p > 0.05$). However, the GGA was significantly better than the hill-climbing (HC) search ($p < 0.001$) and SA ($p < 0.001$).

Table 3.5: Comparison of search heuristics without CFs.

Stand	*GGA*	*SSGA*	*HC*	*SA*
1	14	13.2	9.7	11.3
2	11.7	12.1	10.4	11.3
3	9.1	11.2	7.5	8.8
4	14.1	13.7	12.5	13.5
5	14.4	13.6	12.8	13.7
6	14.9	14.4	12.6	14.7
7	14.1	13.7	12	13.2
8	13.4	12.3	11	12.6
9	12.5	11.7	10.2	9.1
10	12.7	11.7	9.1	11.7
All stands	**130.9**	**127.6**	**107.8**	**119.9**

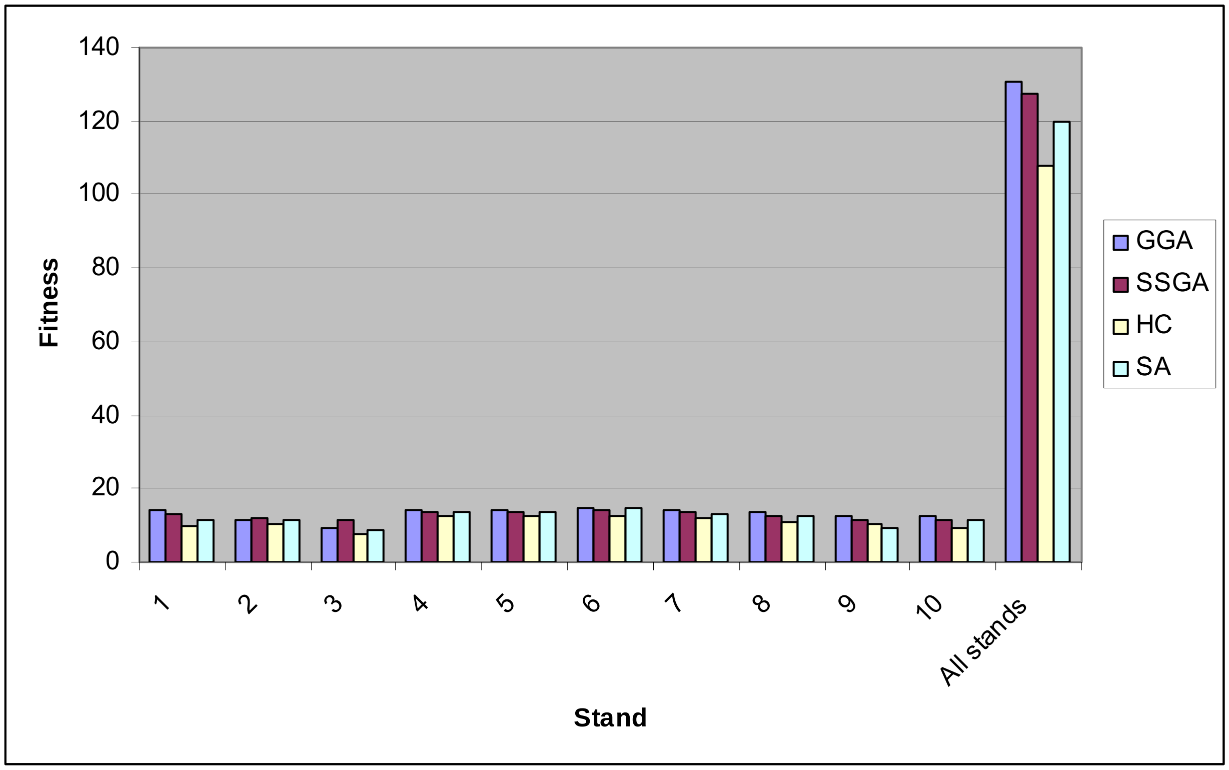


Figure 3.4: Comparison of search heuristics without CFs.

On the comparison of search heuristics with CFs, the search technique used was also significant with a p-value less than 0.001. Just as before, averaging the results from all 10 trials shows that the hill-climbing procedure performed the worst while the GGA turned in the best performance. Table 3.6 shows the best fitness on each condition by stand number. Figure 3.5 presents the same information graphically.

Results on the GGA condition were not significantly different from the SSGA condition (p = 0.066; p > 0.05). However, the GGA performed significantly better than the HC (p < 0.001) and SA (p = 0.002; p < 0.05) conditions.

Essentially the relative search powers of the heuristics were the same whether or not CFs were taken into account. The important point is that the GGA performed as well or better than the other methods under both conditions.

Table 3.6: Comparison of search heuristics with CFs.

Stand	*GGA*	*SSGA*	*HC*	*SA*
1	15.22	14.36	12.72	13.02
2	13.78	13.4	12.98	13.26
3	11.1	12.06	10.32	9.7
4	15.04	14.86	14.32	15.08
5	15.18	14.7	14.14	14.92
6	15.2	15.04	14.72	15.2
7	15.12	14.72	14.14	14.7
8	14.6	14.1	13.04	14.52
9	13.5	12.36	10.84	11.04
10	13.32	12.86	10.94	12.46
All stands	**142.06**	**138.46**	**128.16**	**133.9**

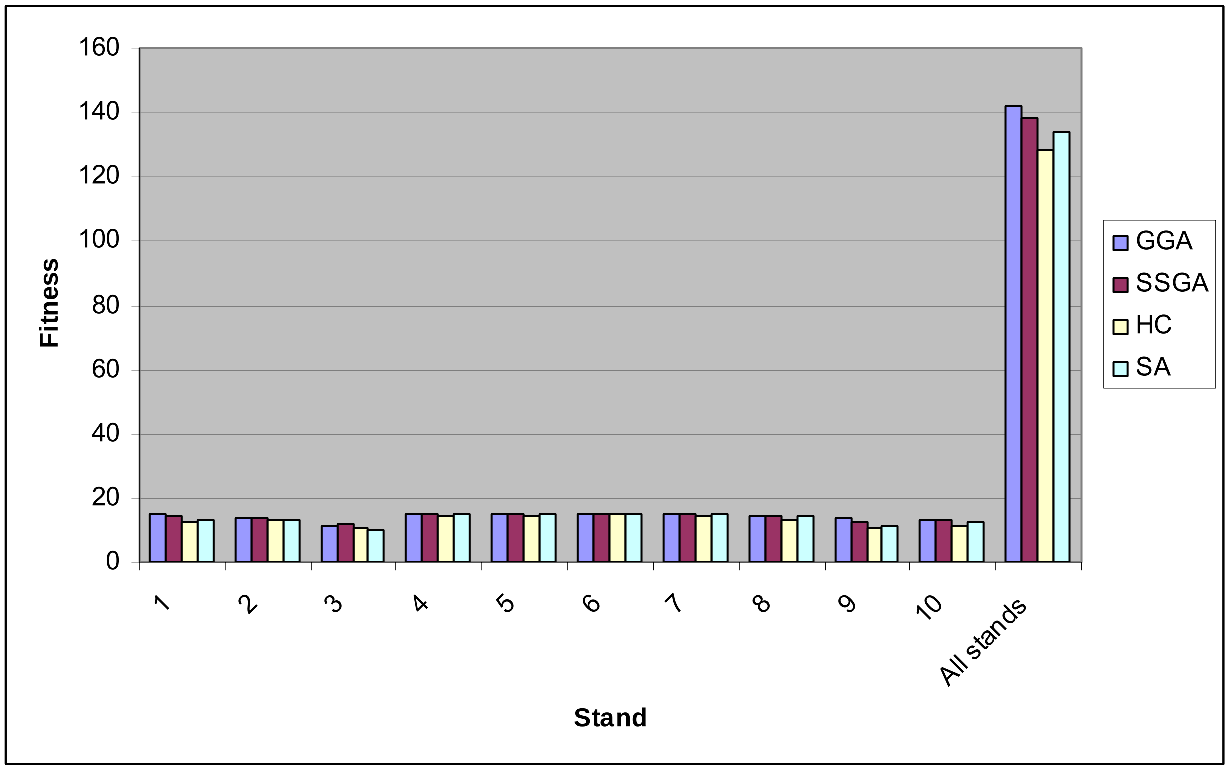


Figure 3.5: Comparison of search heuristics with CFs.

3.5 COMPARING THE GA TO A DOMAIN EXPERT

Lastly plans evolved with the GA were compared to a treatment schedule recommended by a human expert. The expert is a forester who has worked on the NED-2 project and is familiar with the goals and underlying DFCs used by the GA's fitness function. This helped to control for the possibility that the expert might develop a plan based on different values or criteria than the computer generated plans.

3.5.1 GA VS. FORESTRY EXPERT: OVERVIEW AND METHODS

The expert developed a treatment schedule for the Bent Creek experimental forest management unit located near Ashville, North Carolina. A planning period of 40 years beginning in the year 2005 was agreed on, with 5 year intervals between treatments. Because of the difficulty of designing a long term plan with multiple and possibly incompatible objectives, the expert designed a plan for one goal only, periodic income. The periodic income goal emphasizes long-term and sustainable timber production.

At this point in the project it became necessary to reevaluate the DFCs when the domain expert pointed out deficiencies in the existing rules. In previous experiments goals were defined entirely in terms of stand-level DFCs, with the implicit assumption that management-level goals could be satisfied by independently optimizing each stand in the management unit. From a designer's perspective this is an appealing idea because it reduces the complexity of chromosome representations. Instead of representing plans for multiple diverse stands, each chromosome need only represent a plan for one stand. Unfortunately this approach fails for management unit-level goals involving relationships among stands in the unit. For example, if a forest manager is interested in promoting an uneven aged forest with trees of many different sizes, she will need to consider the age and height of trees in each stand compared to other stands in the management unit.

The omission of management unit-level DFCs for timber goals does not invalidate the preceding experiments as measures of the effectiveness of various search parameters, but in order to realistically compare the performance of the GA to a human expert's plan the DFCs for periodic income were expanded to include size class data from multiple stands. This new requirement is a management unit DFC, distinct from the stand DFCs listed in the appendix. A size class is simply a label for a range of DBH values. For example, a "sapling" is a tree with a DBH between one and four inches. The management unit DFCs for periodic income, shown in Figure 3.6, tests the percent of the basal area in a management unit accounted for by each size class. The rule was taken from NED-1, the predecessor of NED-2.

- Percent of area in regeneration >= 5 and <= 10; and
- Percent of area in sapling + Percent of area in pole >= 35 and <=45; and
- Percent of area in small sawtimber >= 25 and <= 35; and
- Percent of area in large sawtimber >= 10 and <= 15; and
- At least 65% of Management Unit Area satisfies the stand DFCs

Figure 3.6: NED-1 management unit DFC for periodic income.

A simple modification of the GA was developed for this experiment that represents individuals as management unit-level plans. Chromosomes in this GA contain a plan for each stand in the management unit. Crossover between individuals is limited to corresponding stands. Put another way, if a management unit plan includes plans for Stand A and Stand B, when crossover occurs between it and another individual the sections corresponding to Stand A crossover independently of the sections corresponding to Stand B. This measure was taken to help guide and constrain search, because plans for different stands are likely to be very different and crossovers between stands may be overly disruptive.

In other respects the management unit GA works very much like the normal version. The fitness of a management unit-level plan is computed by adding the fitness values assigned to each stand-level plan (computed in the usual way), and then adding a special bonus score if the management unit has balanced size classes. The last condition of the NED-1 rule, which requires that 65% of the area satisfies the stand DFCs, was judged to be redundant and left out of the GA. The reasoning behind this decision is that stand-level plans are already assigned fitness values for satisfying stand DFCs, and these values are part of the calculation that determines the fitness of the management unit-level plan. Therefore, plans in which many stands satisfy their DFCs will already be assigned higher fitness values, and do not need to be rewarded twice.

The human expert devised a treatment schedule with a focus on creating a more favorable size-class distribution in the management unit. Bent Creek is a very even-aged and even-sized forest, which is undesirable for a periodic income treatment regime. The expert noted that much of the management unit comprises yellow poplar and oak stands, and devised a treatment schedule to balance size structures with a secondary concern of getting high value species back in the regenerated stands (Rauscher, personal communication, June 11, 2005). Yellow poplar stands were clearcut once during the 40 year planning horizon, and oak stands reduced to a basal area of 60 square feet followed by a clear cut 15 years later. The timing of these treatments was distributed so that, for example, not all the yellow poplar stands were clearcut in the same year.

With 65 stands over 6140 acres, Bent Creek is a much larger management unit that those tested in the other experiments. Nevertheless, it was informally observed during debugging that the GA performed as well or nearly so with a smaller population size than the size 50 used in previous experiments, so the expert's plan was compared to the new management unit-level generational GA using a population size of 40. The cycle length was set to 5, with 9 treatment intervals beginning in 2005 and ending in 2045. The rest of the variable parameters were left at their default values.

In order to test the expert's plan in the same environment as the GA, the plan was first entered into the NED-2 planning interface and simulated. This was necessary to generate FVS keyword files for the plan because it was described in terms of NED-2 treatments. The keyword files created by NED-2 routines were then copied to the GA directory, and the expert plan was evaluated by the GA fitness function. To be safe, the expert and GA plans were also compared by evaluating total removals in terms of merchantable cubic foot volume, sawlog cubic foot volume, and sawlog board foot volume, as well as the amount of harvestable timber left in reserve at the end of the plans. These data were retrieved from the summary statistics table in the FVS *.out file.

Initially the expert's plan fared suspiciously poorly on the GA fitness function. Following a lengthy investigation the problem was traced to the way in which tree class codes were used to calculate volumes of commercially viable timber in a stand. Tree records are assigned a "1", "2", or "3" to indicate that they represent high value, commercially viable, or noncommercial stock respectively. Until this point in the experiments the GA had always relied on FVS to provide these codes. NED-2 provides an alternative means of assigning tree class codes. Each management unit database contains a list of tree species native to the region and their associated values. The tree class codes in the NED-2 databases are more particular to each management unit and therefore probably more accurate. The expert's plan achieved a more reasonable level of success when tested with the NED-2 tree class codes. The GA was also re-tested with the new codes.

3.5.2 GA VS. FORESTRY EXPERT: RESULTS AND DISCUSSION

After running the GA several times a representative sample was chosen to compare to the expert's plan. Neither the best plan evolved with the GA nor the expert's plan resulted in balanced size classes by 2045. Therefore neither plan satisfied the management unit-level DFCs for the periodic income goal. This can be attributed to the very even-aged and even-sized starting conditions of the Bent Creek forest. In terms of stand-level DFCs, the fitness function gave the best solution found by the GA a score of 324.6 while the expert's plan scored 280.8.

The plans were also evaluated by comparing harvest volumes for three categories of timber product: merchantable cubic-foot volume, sawlog cubic-foot volume, and sawlog board-foot volume. The harvest volumes were extracted from the FVS summary output files (*.sum). Tables 3.7 and 3.8 show harvest volumes by year for each plan across the whole management unit, while Tables B.1 and B.2 in Appendix B show harvest volumes by stand number and also provide estimates of unharvested volumes remaining in each stand at the end of the planning horizon.

The plan discovered by the GA resulted in a higher volume for all three categories of timber product compared to the expert's plan, while the expert's plan left more harvestable timber in reserve at the end of the plan. The GA was also somewhat more consistent in the volume of harvest by year, as determined by measuring the variance within each product column over the course of the plans.

Table 3.7: Timber removals by year under forestry expert's plan.

	Product		
Year	merch cu ft	sawlg cu ft	sawlg bd ft
2005	11803	10970	60593
2010	7974	7276	41075
2015	6781	6041	32942
2020	17183	15370	80386
2025	25530	24746	137573
2030	7813	6738	35654
2035	20973	19484	105363
2040	6274	5754	31260
2045	30753	29115	159360
All Years	**135084**	**125494**	**684206**

Table 3.8: Timber removals by year under GA plan.

	Product		
Year	merch cu ft	sawlg cu ft	sawlg bd ft
2005	37385	28906	153083
2010	29499	21036	109360
2015	30110	21143	114553
2020	33018	27273	154941
2025	27702	22522	120232
2030	22935	17758	111579
2035	26926	21204	118194
2040	13384	10159	56225
2045	9342	7025	42058
All Years	**230301**	**177026**	**980225**

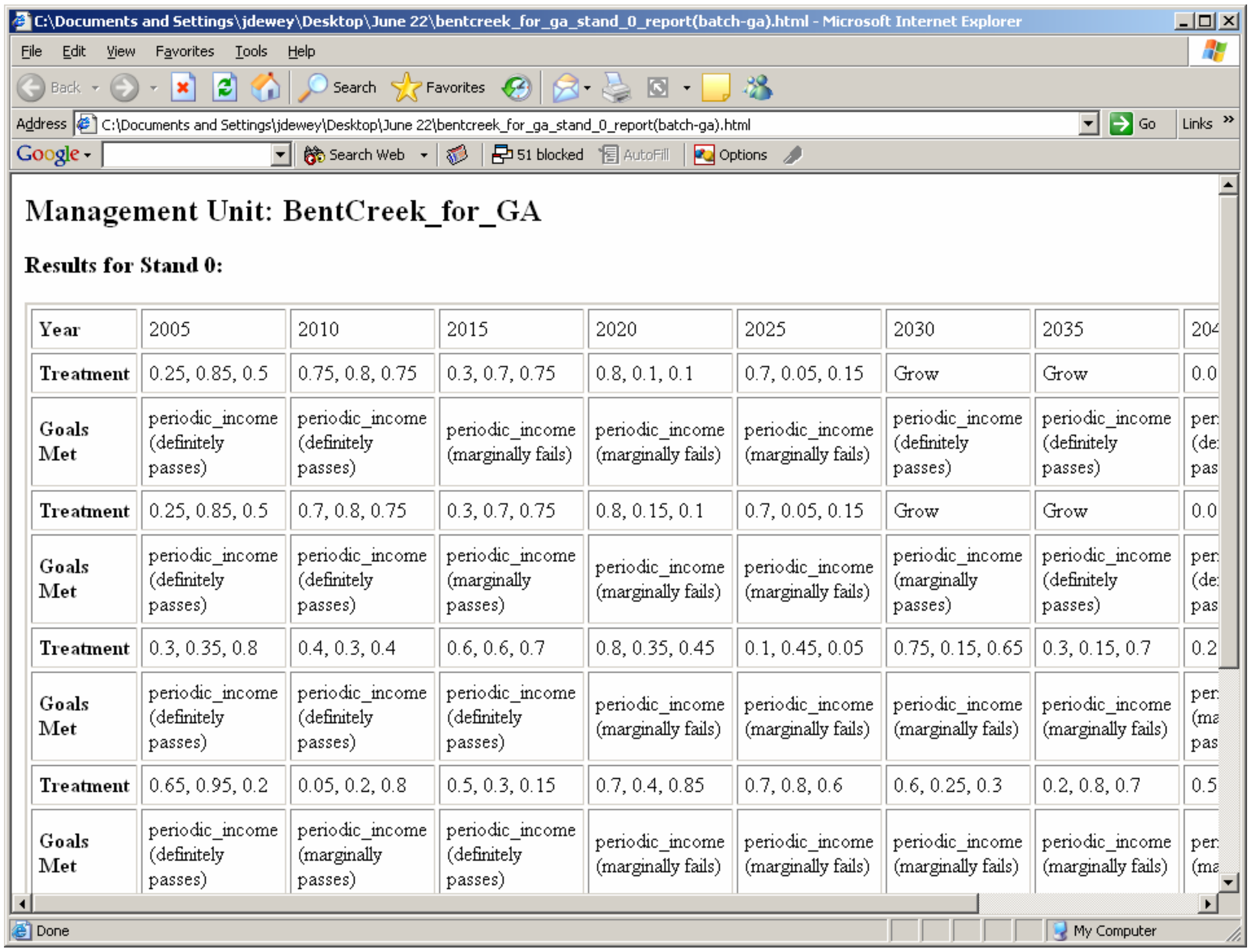
An important caveat to this experiment is that the model of forest regeneration provided with FVS is very rudimentary. This means that certain assumptions made by the expert regarding the course of regeneration following treatments could be correct in reality, but not be modeled by FVS. As such it should be expected that the expert's treatment would underperform in simulation. By contrast, the GA is only as good as its regeneration model. The GA could be upgraded to run other forest growth simulators fairly easily, indicating a possible future avenue for research.

Another limitation to the evaluation methods described here is the lack of an economic model for determining the cost of thinning treatments. Going on raw harvest volumes and stand DFCs alone, the GA appears to surpass the human expert. However, the plan recommended by the GA involves a greater number of treatments than the expert's plan. Clearly, if the goal of a treatment regimen is periodic and sustainable income from timber, the costs associated with performing the treatments need to be outweighed by the profits. At this time of writing, the means for making this determination were not available and therefore not a part of the fitness function's calculations. This would be a desirable addition in any future iterations of the GA program.

CHAPTER 4

PRESENTING RESULTS

Following optimization of each stand, an html report is created using data from the last generation of plans. The report contains a table listing every phenotypically unique management alternative discovered by the GA. Columns represent treatment cycles. Data pertaining to each plan are contained in two rows. The rows labeled “Treatment” contain the plan prescriptions themselves, and the rows underneath show which goals were satisfied during each cycle of the plan. If two or more plans satisfy the same goals in the same treatment intervals, only one appears in the table. Therefore each plan represents a distinct management alternative. Presentation is further simplified by predicates that replace treatments interpreted as doing nothing or clearcutting in the table with the words “grow” and “clearcut” respectively.


C:\Documents and Settings\jdewey\Desktop\June 22\bentcreek_for_ga_stand_0_report(batch-ga).html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Documents and Settings\jdewey\Desktop\June 22\bentcreek_for_ga_stand_0_report(batch-ga).html
Go Links
Google Search Web 51 blocked AutoFill Options
Done
My Computer

Management Unit: BentCreek_for_GA

Results for Stand 0:

Year	2005	2010	2015	2020	2025	2030	2035	204
Treatment	0.25, 0.85, 0.5	0.75, 0.8, 0.75	0.3, 0.7, 0.75	0.8, 0.1, 0.1	0.7, 0.05, 0.15	Grow	Grow	0.0
Goals Met	periodic_income (definitely passes)	periodic_income (definitely passes)	periodic_income (marginally fails)	periodic_income (marginally fails)	periodic_income (marginally fails)	periodic_income (definitely passes)	periodic_income (definitely passes)	per (de pas
Treatment	0.25, 0.85, 0.5	0.7, 0.8, 0.75	0.3, 0.7, 0.75	0.8, 0.15, 0.1	0.7, 0.05, 0.15	Grow	Grow	0.0
Goals Met	periodic_income (definitely passes)	periodic_income (definitely passes)	periodic_income (marginally passes)	periodic_income (marginally fails)	periodic_income (marginally fails)	periodic_income (marginally passes)	periodic_income (definitely passes)	per (de pas
Treatment	0.3, 0.35, 0.8	0.4, 0.3, 0.4	0.6, 0.6, 0.7	0.8, 0.35, 0.45	0.1, 0.45, 0.05	0.75, 0.15, 0.65	0.3, 0.15, 0.7	0.2
Goals Met	periodic_income (definitely passes)	periodic_income (definitely passes)	periodic_income (definitely passes)	periodic_income (marginally fails)	periodic_income (marginally fails)	periodic_income (marginally fails)	periodic_income (marginally fails)	per (ma pas
Treatment	0.65, 0.95, 0.2	0.05, 0.2, 0.8	0.5, 0.3, 0.15	0.7, 0.4, 0.85	0.7, 0.8, 0.6	0.6, 0.25, 0.3	0.2, 0.8, 0.7	0.5
Goals Met	periodic_income (definitely passes)	periodic_income (marginally passes)	periodic_income (definitely passes)	periodic_income (marginally fails)	periodic_income (marginally fails)	periodic_income (marginally fails)	periodic_income (marginally fails)	per (ma

Figure 4.1: Example of HTML stand report.

CHAPTER 5
PROGRAM EXECUTION

This section briefly describes the execution of the main generational GA program, which optimizes stands independently. The program is too large to describe in precise detail, but descriptions are provided for the most important predicates and presented in the general order by which they are called when the program is run. All code for the GA is implemented with LPA WIN-Prolog 4300.

Before the GA program begins there are several one-place predicates defining general parameters that may be adjusted. The predicate **database/1** tells the program which directory to look in for tree inventory input data (*.tin) and computed variable data (*.pl) files. These files are necessary to create the keyword file needed by FVS to run simulations and must be generated before executing the GA using the NED-2 predicate **mdb2fvs/0**. Glende (2004) describes the operation of this predicate in detail. Figure 5.1 provides an overview of dataflow in the GA.

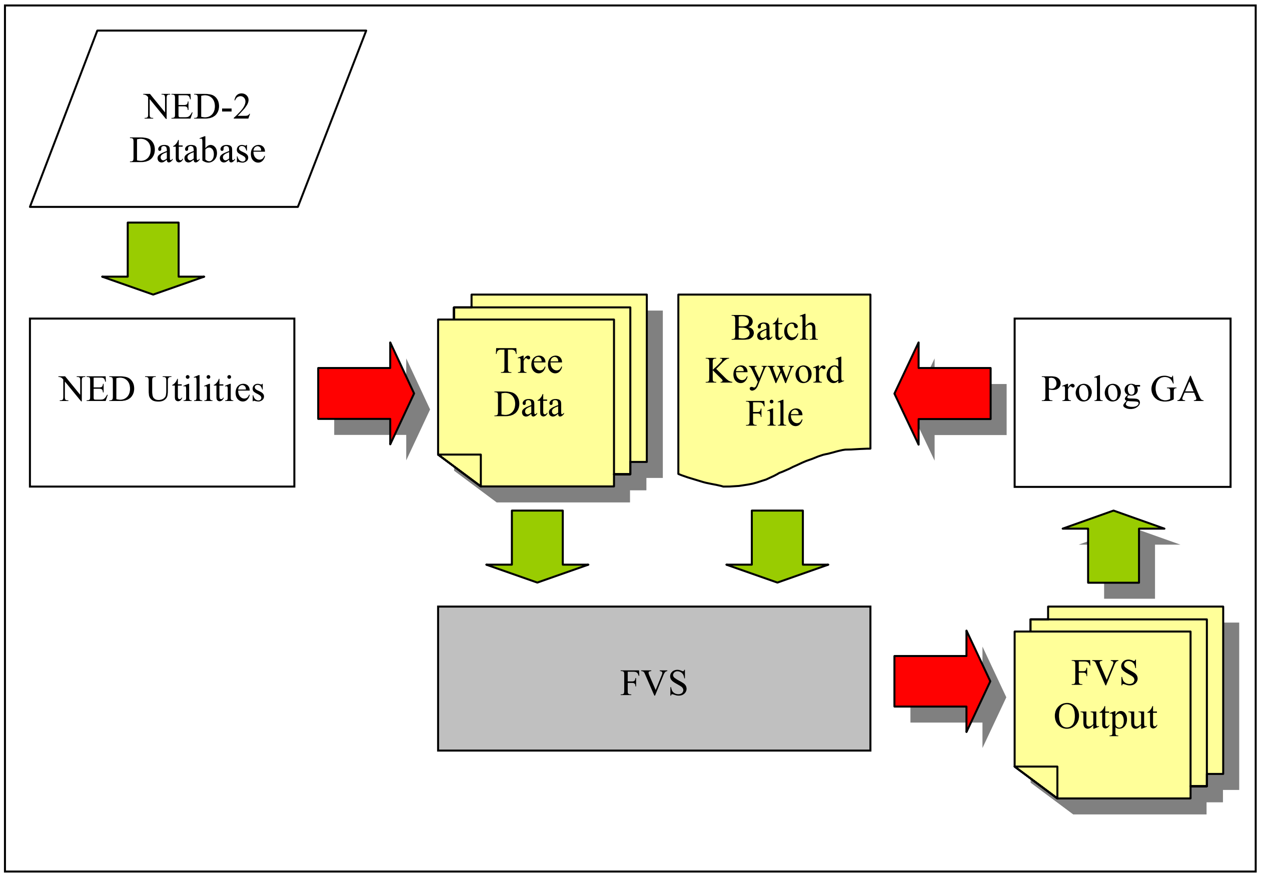
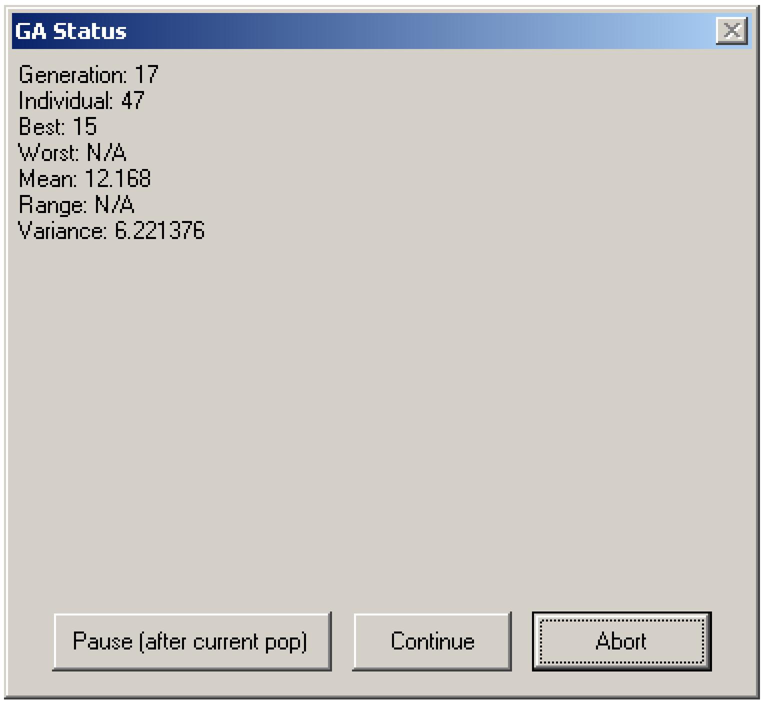


Figure 5.1: Dataflow in the GA.

Other initialization parameters tell the program which stands from the database to simulate (**stand_list/1**), how large the population should be (**population_size/1**), how many generations to run (**max_generation/1**), the probability of mutation (**mutation/1**) crossover rate (**crossover_rate/1**), the type of fitness function (**ff_type/1**), the selection method (**selection/1**), the number and length of treatment cycles in a treatment schedule (**fvs_number_cycles/1** and **fvs_cycle_length/1**), the number of most fit individuals to carry forward automatically (**elitism/1**), and whether or not to enforce genotypic uniqueness (**enforce_genotypic_uniqueness/1**).

When the parameters are set to the desired values and the program is compiled, the user is prompted to begin by typing "go." The **go/0** predicate begins execution of the program. First the program loads needed files and tree inventory data whose location is specified by the **database/1** parameter. A windows dialog box is created that will continue to be updated over the course of the GA run, displaying the current individual number being processed, the current generation number, the best solution found so far (for the current generation), the worst fitness, and mean fitness and the standard deviation, as shown in Figure 5.2. Some of this information may not be available or applicable to all variations of the GA program. The user may exit the program at any time by clicking the 'Abort' button.



```
Generation: 17
Individual: 47
Best: 15
Worst: N/A
Mean: 12.168
Range: N/A
Variance: 6.221376
```

Figure 5.2: GA Status dialog box.

Next the program enters the recursive predicate **mu_loop/4**, which performs GA optimization on each stand in **stand_list/1** one by one. First the **init/0** predicate is queried, which cleans up old files left over from the optimization of any previous stands. Then the predicate **new_mu_population/3** creates a random initial population from scratch for the current stand. Each individual plan is assigned an uninstantiated variable for its fitness and a unique number *N*, where *N* is an integer in the range from zero to the size of the population. Information about the starting population is asserted, and the **run_generations/4** predicate is queried, which runs the GA for the given number of cycles. The **run_generations/4** predicate is the recursive heart of the program and is shown in Figure 5.3.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A LOOP TO RUN THE GA FOR THE GIVEN NUMBER OF CYCLES.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% At end, hide dialog.

run_generations(AverageFitnessStack,BestFitnessStack,GeneDifferenceStack,
      UniqueStack) :-

      % Stop if flag is set to 1,
      switch(f, Stop),
      Stop == 0,

      % Get the population
      generation_data([CurrentPop, Stand, Snapshot, TreeFile, PopSize,
      Cycles]),

      % Update generation number
      generation(GOld),
      GNew is GOld + 1,

      % Initialize the best and worst fitness values
      retractall(current_best(_,_, _)),
      assert(current_best(0,0,0)),
      retractall(current_worst(_,_, _)),
      assert(current_worst(0,0,99999999)),

      % Create a batch keyword file for the whole population
      write_big_keyfile0(CurrentPop, Stand, Snapshot,TreeFile),

      % Process one generation
      run_generation(CurrentPop, Stand, Snapshot, TreeFile),

      % Sort the population by fitness
      findall(    [IndFitness, IndNum, Indi, Parents],
                  member([IndFitness,  IndNum, Indi, Parents], CurrentPop),
                  IndListTemp),
      sort(IndListTemp,IndList),


      % A test--get number of unique individuals in pop
      findall(TP,member([_,_,TP,_],IndList),MyList),
      sort(MyList,MySorted),
      length(MySorted,Unique),

      % A test--checks genotypic distance between
      % best and worst plans in current pop
      first_element(IndList,[_,_,WorstPlan,_]),
      last_element(IndList,[BestFitness,_,BestPlan,_]),
      list_distance(BestPlan,WorstPlan,DifferenceList),
      sum(DifferenceList,Difference),
```

Figure 5.3: The recursive predicate **run_generations/4**.

```
      % Compile a list of fitness values
      findall(Val, member([Val,_, _,_],IndList), FitValList),

      % Update statistics
      update_stats(Cycles , FitValList),
      update_dialog_alt(GNew, 1,FitValList),
      current_mean(Avg),

      % Create the next population/generation
      generate_new_pop(IndList, FitValList, PopSize, PopTemp),

      %Save the old population if specified
      save_pop(CurrentPop),

      % add uninstantiated fitness variables to new population
      findall(    [FitVal, IndNum,Ind,Parents2],
                  member([Ind,Parents2], PopTemp,IndNum),
                  NewPopulation),
      !,

      % if at end then stop,
      (
      Cycles < 2,
      last_pop(CurrentPop),

      % set the stop flag to true
      switch(f, 1),

      % Print some data directly to the console (for diagnostic purposes)
      print_results([Avg|AverageFitnessStack],[BestFitness|BestFitnessStack
      ],[Difference|GeneDifferenceStack],[Unique|UniqueStack]),
      !
      ;
      % ..else run the next population.

      NextCycle is Cycles -1,

      % Assert the newly spawned generation as the current population
      retractall(generation_data(_)),
      assert(generation_data([NewPopulation, Stand, Snapshot,
      TreeFile,PopSize, NextCycle])),
      !,

      % Recursive call to run_generations/4
      run_generations([Avg|AverageFitnessStack],[BestFitness|BestFitnessSta
      ck],[Difference|GeneDifferenceStack],[Unique|UniqueStack])
      ).

% At the end, hide the dialog box
run_generations(_,_,_,_) :-
      wshow(new_dialog, 0).
```

Figure 5.3 (Cont): The recursive predicate **run_generations/4**.

The **run_generations/4** predicate uses a **switch/2** statement to loop until a counter that keeps track of the number of generations simulated so far reaches its stopping point. As long as the switch has not been triggered the program goes forward with processing the population for another generation. First a keyword file (*.key) defining the population in terms understandable by FVS is created with the **write_big_keyfile/4** predicate. Writing a keyword file for the entire population allows the program to run in batch mode, performing all the simulations for a generation in one sweep. This saves times compared with starting and closing FVS separately for each individual in the population. After the keyword file is written **run_generation/4** is queried, which simulates treatments in the population for a single generation.

Inside of **run_generation/4** the program gets input data from the tree data file (*.tin) and keyword file and runs the FVS simulation. The results for each stand are written to the tree list output file (*.out), with the stand number in the name for identification. The output files contain all the information necessary for evaluating the fitness of each treatment in the population. Before this happens however the data must undergo additional processing. Data from the output files is processed differently depending on which fitness function is selected. The baseline fitness function used in most of the testing uses the predicate **process_output3/3** to read the batch data for each individual in the population recursively and calls **dfc_fitness/3**, which assigns fitness.

Within the **dfc_fitness/3** predicate, relevant tree data read from the tree list file is extracted. These data are the species code, trees per acre, DBH, and a code indicating the timber quality. The timber quality may be classified as cull (no timber value), regular, or high value. Other values needed in fitness calculations such as relative density and basal area are estimated from the DBH and basal area using formulas borrowed from expert knowledge incorporated into NED-2. The **dfc_fitness_by_year/5** predicate gets the fitness for each treatment interval, or cycle, while keeping a record of which goals were satisfied in each cycle. The goal analysis for each year is performed by a call to **check_goals/3**, which sums the values indicating the degrees with which the DFCs for each goal are satisfied. The fitness values from each treatment interval in a plan are summed to get the fitness of the entire plan.

After every plan in the population has been assigned a fitness value **generate_new_pop/4** creates the next generation from the current population. First the number of "best" solutions to be automatically passed (*N*) on is instantiated with the **elitism/1** predicate. Then, **best_n/3** finds the best *N* solutions in the current population and puts them in a stack: This is the starting point for the new generation. The rest of the new population is filled in by checking the selection operator (**selection_type/1**) and applying it to select the best from the old population using **Selection/5**. Next the old population is saved if specified (**save_pop/1**), and uninstantiated fitness values are added to the new population. If there are no more cycles left to run, the GA changes a **switch/2** value that will cause the program to exit from the run_generations/4 segment and prints the results of the search to the console. Otherwise, the current generation data is asserted to working memory, the number of cycles left to run is decremented by 1, and the next generation is simulated by **run_generations/4**.

When every stand has been optimized by run_generations/4, the program exits from **mu_loop_aux/4** back into the **mu_loop/4**. If the GA has run its entire course the number of times specified by **run_this_many_times/1**, the program is finished. Figure 5.4 summarizes the control hierarchy of the most important predicates in an outline.

- **mu_loop/4** Optimize fitness for each stand
    - **run_generations/4** A loop to run the GA for a given # of cycles
        - **write_big_keyfile0/4** Create keyword file for whole population
        - **run_generation/4** Simulate treatments for one generation
            - **fvs_run/4** Simulate plans
            - **process_output3/3** Assign fitness to plans
        - **generate_new_pop/4** Create the next generation
            - **best_n/3** Most fit plans advance automatically
            - **rank/5** Use rank-based selection

Figure 5.4: GA predicate hierarchy.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

In the current study the standard generational GA fared well against other heuristics such as simulated annealing and the steady state GA, and by most measures produced better results in simulation than a human expert's recommendation. Promising results in the latter experiment must be qualified by noting a relative lack of sophistication in the FVS regeneration model compared to other models of forest growth. A further reservation involves the large number of relatively minor treatments recommended by the GA, compared to the smaller number of treatments in the expert's plan. Although following a plan recommended by the GA would result in a steady flow of merchantable timber, the costs associated with actually performing the needed treatments was not factored into the fitness function. A more detailed economic model of the costs and gains associated with various treatments is still needed if this program is to serve a practical prescriptive function.

Reservations aside, the GA approach to the problem of treatment prescription is promising for a couple of reasons. First, the search space (number of possible solutions) for a large management unit is too big to test every possibility in a reasonable amount of time. There may be general heuristics that could be helpful in making plans, but these would likely be of limited use because of geographic differences in forest types around the country. Using a GA is an attractive alternative to relying on exhaustive search techniques or general rules of thumb that may not always be applicable. Although GAs are not guaranteed to find optimal solutions (De

Jong, 1993), when the search space is sampled effectively they often find very good solutions and sometimes even the optimal solution (Goldberg 1989).

Future developments of the GA may expand the goal analysis to include more non-timber goals and a realistic economic model. Another limitation of the current program is the lack of a planting treatment. This could be important if, for example, the only way to satisfy a timber goal is to introduce a new, commercially valuable species to the stand. Current treatment definitions are also limited by the fact that they are applied to all species. Specifying that certain species should not be cut would complicate the representation of treatments, but more accurately reflect real life management options.

Additional care also needs be taken to present the results of the GA search in terms easily understandable by someone unfamiliar with the inner workings of the program. Currently the HTML stand reports generated by the program display treatments as they are represented in the GA. These values could be easily transformed into values more comprehensible to a forest manager, such as the minimum and maximum DBH values of the trees to be cut, in the same way that they are interpreted by FVS before simulation. The incorporation of additional simulators as alternatives to FVS would also be a worthwhile addition to the program. With these improvements the GA could be a powerful and practical prescriptive tool.

REFERENCES

Bentley, P. J., & Wakefield, J. P. (1997). Finding acceptable solutions in the Pareto-optimal range using multiobjective genetic algorithms. In P. K. Chawdhry, R. Roy, & R. K Pant (Eds.), *Soft Computing in Engineering Design and Manufacturing, Part 5* (pp. 231-240). London, UK: Springer-Verlag.

Bettinger, P., & Graetz, D. (2003, October). *Determining thinning regimes to reach stand density targets for any-aged stand management in the Blue Mountains of eastern Oregon*. Systems Analysis in Forest Resources: Proceedings of the 2003 Symposium, Stevenson, WA.

Boston, K., & Bettinger, P. (2002). Combining Tabu Search and Genetic Algorithm Heuristic Techniques to Solve Spatial Harvest Scheduling Problems. *Forest Science 48* (1), 35-46.

Chafekar, D., Xuan, J., & Rasheed, K. (2003). Constrained multi-objective optimization using steady state genetic algorithms. *Proceedings of the 2003 Genetic and Evolutionary Computation Conference*. Retrieved May 16, 2005 from http://www.cs.uga.edu/~khaled/papers385.pdf

De Jong, K. A. (1993). Genetic algorithms are NOT function optimizers. In: Whitley, D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 5-17). San Mateo, CA: Morgan Kaufmann.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. New York: John Wiley & Sons.

Deb, K., & Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex Systems 9*(2), 115-148.

Deb, K., & Kumar, A. (1995). Real-coded genetic algorithms with simulated binary crossover: Studies on multi-modal and multi-objective problems. *Complex Systems 9*(6), 431-454.

Dixon, G. E. (2002). *Essential FVS: A user's guide to the Forest Vegetation Simulator*. Internal Rep. Fort Collins, CO: U. S. Department of Agriculture, Forest Service, Forest Management Service Center.

Ducheyne, E. I., De Wulf, R. R., & Baets, B. D. (2001). Bi-objective genetic algorithms for forest management: a comparative study. In Spector et al. (Eds.) *Proceedings of the 2001 Genetic and Evolutionary Computation Conference. Late-Breaking Papers* (pp. 63-66). San Francisco: Morgan Kaufmann.

Eshelman, L. J., & Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In L. Whitley (Ed.), *Foundations of Genetic Algorithms 2* (pp. 187-202). Los Altos, CA: Morgan Kaufmann.

Feng, C., & Lin, J. (1999). Using a genetic algorithm to generate alternative sketch maps for urban planning. *Computers Environment and Urban Systems, 23*, 91-108.

Glende, A. (2004). *The NED forest management DSS: The integration of growth and yield models*. Unpublished masters thesis, the University of Georgia, Athens, Georgia.

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

Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems 5*(2), 139-168.

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems 4*(4), 415-444.

Gong, P. (1992). Multiobjective dynamic programming for forest resource management. *Forest Ecology and Management, 48*, 43-54.

Hamming, R. W. (1980). *Coding and information theory*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Herrera, F., Lozano, M., & Verdegay, J. L. (1998). Tracking real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review 12*(4), 265-319.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems* (2nd ed.). Cambridge: MIT Press.

Holsapple, C. W., & Whinston, A. B. (1996). *Decision support systems: A knowledge-based approach*. Minneapolis/St. Paul, MN: West Publishing.

Hughell, D. A., & Roise, J. P. (1997, May). *Simulated adaptive management for timber and wildlife under uncertainty*. Proceedings of the 1997 Symposium on Systems Analysis in Forest Resources, Traverse City, Michigan.

Luke, B. T. (2005). *Simulated annealing*. Retrieved June 8, 2005, from http://www.cs.sandia.gov/opt/survey/sa.html

Matthews, K. B., Sibbald, A. R., & Craw, S. (1999). Implementation of a spatial decision support system for rural land use planning: integrating geographic information system and environmental models with search and optimisation algorithms. *Computers and Electronics in Agriculture 23*, 9-26.

Morrison, J. (1993). Integrating ecosystem management and the forest planning process. In M. Jensen, & P. Bourgeron (Eds.), *Eastside Forest Ecosystem Health Assessment: Vol 2. Ecosystem Management: Principles and Applications*. (pp. 281-290). USDA Forest Service.

Mullen, D. S., & Butler, R. M. (1999). The design of a genetic algorithm based spatially constrained timber harvest scheduling model. In J. Vasievich et al. (Eds.), *Proceedings of the Seventh Symposium on Systems Analysis in Forest Resources*, May 28-31, 1997, Traverse City, Michigan. USDA Forest Services North Central Experiment Report.-205. pp.57-65.

Nute, D., Rosenberg, G., Nath, S. Verma, B, Rauscher, H. M., & Twery, M. J. (2000). Goals and goal orientation in decision support systems for ecosystem management. *Computers and Electronics in Agriculture, 27*, 357-377.

Nute, D. E., Potter, W. D., Maier, F., Wang, J., Twery, M. J., Rauscher, H. M., Knopp, P., Thomasma, S. A., Dass, M., Uchiyama, H., & Glende, A. (2004). NED-2: An agent-based decision support system for forest ecosystem management. *Environmental Modeling and Software 19*, 831-843.

Radcliffe, N. J. (1992). Non-linear genetic representations. In R. Manner, & B. Manderick (Eds.), *Parallel Problem Solving from Nature*, *2*, (pp. 259-268). Amsterdam: Elseviar.

Rauscher, H.M., Lloyd, F.T., Loftis, D.L., & Twery, M.J. (2000). A practical decision-analysis process for forest ecosystem management. *Computers and Electronics in Agriculture, 27*, 195-226.

Reeves, C. R. (1993). Using genetic algorithms with small populations. In S. Forrest (Ed.) *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 92-99). San Mateo, CA: Morgan Kaufman.

Routh, C. (2004). *The NED-2 forest ecosystem management DSS: The integration of wildfire risk and GIS agents*. Unpublished masters thesis, the University of Georgia, Athens, Georgia.

Schaffer, J.D. (1984). Some experiments in machine learning using vector evaluated genetic algorithms. Ph.D. thesis, Vanderbilt University, Nashville, Tennessee.

Smith, R. E. (1994). Genetic and evolutionary systems. *Adaptive Computing: Mathematics, Electronics, and Optics CR55*, 151-174.

Thorsen, B. J., & Helles, F. (1998). Optimal stand management with endogenous risk of sudden destruction. *Forest Ecology and Management 108*, 287-299.

Whitley, Darrell. (1989) The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J.D. Shaffer (Ed.) *Proceedings of the third international conference on genetic algorithms* (pp.133-140). San Mateo, CA: Morgan Kaufman.

Wright, A. (1991). Genetic algorithms for real-parameter optimization. In J. Rawlkins (Ed.), *Foundations of Genetic Algorithms 1* (pp. 205-218). San Mateo, CA: Morgan Kaufman.

APPENDIX A

QUICK REFERENCE: GOALS AND DFCS

Figures A-1 through A-5 contain descriptions of five management goals in terms of desired future conditions (DFCs). The goals are borrowed from NED-1 and -2 and re-implemented in the genetic algorithm program. The information in this appendix is also available in the NED-1 help file.

Focus on Periodic Income

Goal Description

The landowner desires to maximize periodic (or annual) income -- usually by favoring high-value products.

Desired Future Conditions (DFCs)

To achieve this goal, the following DFCs must be met:

Management unit level

· Percent of area in regeneration >= 5 and <= 10; and
· Percent of area in sapling + Percent of area in pole >= 35 and <=45; and
· Percent of area in small sawtimber >= 25 and <= 35; and
· Percent of area in large sawtimber >= 10 and <= 15; and
· At least 65% of Management Unit Area satisfies the stand DFCs

Stand level – The following table lists DFCs by Prescription forest type.

	Relative density		Basal area		Basal area of AGS
	>=	<	>=	<	>=
Allegheny hardwoods	60	100			30
Appalachian hardwoods	60	100			30
aspen - birch			60	140	30
hemlock - hardwoods	60	100			30
northern hardwoods	60	100			35
oak - hickory	60	100			30
oak - northern hardwoods	60	100			30
spruce - fir			80	160	35
spruce - hardwoods			80	140	35
white pine	60	100			35

Figure A-1: Focus on periodic income.

Focus on Cubic-Foot Production

Goal Description

The landowner desires to maximize cubic volume yield - usually to favor timber products such as pulpwood and other fiber products.

Note: In the Allegheny hardwoods forest type, the values of sawtimber products are so high that landowners may prefer to manage for board-foot production and use the cubic-foot by-products of thinnings to meet cubic-foot objectives.

Desired Future Conditions (DFCs)

To achieve this goal, the following DFCs must be met:

Management unit level

- Percent of area in regeneration >= 5 and <= 10; and
- Percent of area in sapling + Percent of area in pole >= 35 and <=45; and
- Percent of area in small sawtimber >= 25 and <= 35; and
- Percent of area in large sawtimber >= 10 and <= 15; and
- At least 65% of Management Unit Area satisfies the stand DFCs

Stand level – The following table lists DFCs by Prescription forest type

	Relative density		Basal area		Basal area of AGS
	>=	<	>=	<	>=
Allegheny hardwoods	60	100			50
Appalachian hardwoods	60	100			30
aspen - birch			60	140	30
hemlock - hardwoods	60	100			30
northern hardwoods	60	100			35
oak - hickory	60	100			30
oak - northern hardwoods	60	100			30
spruce - fir			80	140	35
spruce - hardwoods			80	140	35
white pine	60	100			35

Figure A-2: Focus on cubic-foot production.

Focus on Board-Foot Production

Goal Description

The landowner desires to maximize board-volume yield -- usually to favor timber products such as sawtimber and veneer, or other high-value products.

Desired Future Conditions (DFCs)

To achieve this goal, the following DFCs must be met:

Management unit level

- Percent of area in regeneration >= 5 and <= 10; and
- Percent of area in sapling + Percent of area in pole >= 35 and <=45; and
- Percent of area in small sawtimber >= 25 and <= 35; and
- Percent of area in large sawtimber >= 10 and <= 15; and
- At least 65% of Management Unit Area satisfies the stand DFCs

Stand level - The following table lists DFCs by Prescription forest type.

	Relative density		Basal area		Basal area of AGS	% BA High value spp	% BA comm spp
	>=	<	>=	<	>=	>=	>=
Allegheny hardwoods	60	100			30	25	85
Appalachian hardwoods	60	100			30	25	85
aspen - birch			60	140	30	25	85
hemlock - hardwoods	60	100			30	25	85
northern hardwoods	60	100			35	25	85
oak - hickory	60	100			30	25	85
oak - northern hardwoods	60	100			30	25	85
spruce - fir			80	160	35	25	85
spruce - hardwoods			80	140	35	25	85
white pine	60	100			35	25	85

Figure A-3: Focus on board-foot production.

Focus on Net Present Value

Goal Description

The landowner desires to maximize net present value -- treating the forest property in the same manner as any other investment.

Desired Future Conditions (DFCs)

To achieve this goal, the following DFCs must be met:

Management unit level

- Percent of area in regeneration >= 5 and <= 10; and
- Percent of area in sapling + Percent of area in pole >= 35 and <=45; and
- Percent of area in small sawtimber >= 25 and <= 35; and
- Percent of area in large sawtimber >= 10 and <= 15; and
- At least 65% of Management Unit Area satisfies the stand DFCs

Stand level - The following table lists DFCs by Prescription forest type.

	Relative density		Basal area		Basal area of AGS	% BA High value spp	% BA comm spp
	>=	<	>=	<	>=	>=	>=
Allegheny hardwoods	60	100			30	25	85
Appalachian hardwoods	60	100			30	25	85
aspen - birch			60	140	30	25	85
hemlock - hardwoods	60	100			30	25	85
northern hardwoods	60	100			35	25	85
oak - hickory	60	100			30	25	85
oak - northern hardwoods	60	100			30	25	85
spruce - fir			80	160	35	25	85
spruce - hardwoods			80	140	35	25	85
white pine	60	100			35	25	85

Figure A-4: Focus on net present value.

Enhance Big Tree Appearance

<u>Goal Description</u>

The landowner desires to create or hasten the development of an old-growth or large tree appearance.

<u>Desired Future Conditions (DFCs)</u>

To achieve this goal, the following DFCs must be met:

Stand level:

· Stems per unit area in saplings <= 1000 Stems per acre in saplings; and
· Number of big trees per unit area >= 30 trees per acre

Figure A-5: Enhance big tree appearance.

APPENDIX B

TIMBER REMOVALS BY STAND

Tables B-1 and B-2 display volumes of timber removals by stand number on schedules recommended by a forestry expert and the genetic algorithm respectively. Timber volumes are listed for three categories: merchantable cubic foot volume, sawlog cubic foot volume, and sawlog board foot volume.

Table B-1: Timber removals by stand under forestry expert's plan.

	Total Removals			***Timber remaining in 2046***		
Stand	merch cu ft	sawlg cu ft	sawlg bd ft	merch cu ft	sawlg cu ft	sawlg bd ft
0	0	0	0	4724	4002	20646
1	0	0	0	4796	4371	22415
2	5074	4171	21356	0	0	0
3	0	0	0	7955	7266	55597
4	0	0	0	7953	7264	55544
5	7626	7796	43636	0	0	0
6	0	0	0	5972	4615	23917
7	0	0	0	5344	4854	25335
8	0	0	0	4612	4011	21987
9	0	0	0	5324	2976	15845
10	0	0	0	4681	4186	22689
11	0	0	0	6771	4964	34735
12	0	0	0	5306	2952	15691
13	0	0	0	4804	4183	22217
14	0	0	0	6569	5752	27901
15	0	0	0	5157	2832	15254
16	6314	6453	35978	203	0	0
17	3180	2865	15607	2021	1860	10129
18	6689	6768	38521	78	0	0
19	5565	4734	26157	7	0	0
20	8353	7860	41108	0	0	0
21	4129	3461	18489	578	0	0
22	0	0	0	4102	3723	20612
23	3048	2750	15239	1932	1781	9862
24	3585	3146	16932	311	0	0
25	4664	3857	18955	0	0	0
26	4071	4195	24560	820	193	871
27	3136	2882	15089	225	0	0
28	0	0	0	4943	4276	22129
29	0	0	0	5927	5859	32645
30	0	0	0	5051	4251	22153
31	3970	3719	18702	502	0	0
32	2530	2301	12204	1764	1608	8538
33	0	0	0	5165	4654	25350
34	0	0	0	5943	5879	32809
35	3174	2859	15568	2021	1859	10119
36	2838	2471	13398	2305	2126	11614
37	0	0	0	4664	4245	22490
38	0	0	0	5938	5873	32759
39	0	0	0	4635	4176	22054
40	0	0	0	6976	5996	29795

Table B-1 (Cont): Timber removals by stand under forestry expert's plan.

	Total Removals			***Timber remaining in 2046***		
Stand	merch cu ft	sawlg cu ft	sawlg bd ft	merch cu ft	sawlg cu ft	sawlg bd ft
41	3181	2795	15348	2151	1935	10680
42	0	0	0	4640	4182	22102
43	2818	2452	13277	2299	2118	11554
44	0	0	0	5264	3804	20420
45	2836	2470	13388	2304	2125	11607
46	0	0	0	4643	4185	22124
47	4442	3942	23658	33	0	0
48	5918	5848	32555	0	0	0
49	6584	6674	36857	1533	0	0
50	0	0	0	4345	3956	20954
51	4484	3914	19907	0	0	0
52	2342	2108	11521	0	0	0
53	4580	4062	24198	154	0	0
54	3602	3237	17736	318	0	0
55	3841	3489	17875	255	0	0
56	4230	3769	19783	0	0	0
57	0	0	0	5908	5837	32457
58	0	0	0	5545	3793	17886
59	0	0	0	4650	4214	22291
60	8280	8446	46604	0	0	0
61	0	0	0	5194	4684	25572
62	0	0	0	5367	4882	25540
63	0	0	0	8368	7670	58205
64	0	0	0	5371	4887	25567
All Stands	**261872**	**243256**	**1329492**	**391018**	**333819**	**1870107**

Table B-2: Timber removals by stand under GA plan.

	Total Removals			***Timber remaining in 2046***		
Stand	merch cu ft	sawlg cu ft	sawlg bd ft	merch cu ft	sawlg cu ft	sawlg bd ft
0	2825	1882	9343	1236	1213	6830
1	2712	1789	8662	1626	1208	6885
2	3392	2708	13302	1656	1124	6226
3	7547	6857	48833	2283	2131	16556
4	8204	7481	55371	994	774	5501
5	6405	6399	34613	1775	1697	10353
6	4620	2003	10067	3198	493	2469
7	2272	1853	9100	3228	2992	15910
8	1070	252	1266	2859	2763	15862
9	3212	1848	9463	3064	16	89
10	1605	668	3304	3179	3077	17371
11	4645	1905	12141	1822	1724	15080
12	3801	1255	6534	747	329	1980
13	3489	2857	14950	1596	1225	6832
14	1177	239	1084	5391	5360	27075
15	1993	723	3892	2999	189	1109
16	2956	2913	15359	4186	4194	25885
17	4068	3257	17454	528	329	1989
18	848	823	4173	6306	6461	38383
19	4388	3652	20028	1438	1108	5630
20	7565	7146	36425	870	334	1525
21	3429	3006	15094	1652	1591	9375
22	2946	2200	12120	1142	981	5789
23	3984	3305	17962	591	535	3125
24	3076	2567	13683	1177	1072	6379
25	3597	2527	11926	1133	893	4810
26	4416	4335	25493	471	155	967
27	2127	1845	9516	1462	1369	7676
28	2642	1442	7025	1941	1546	8727
29	3785	2408	12910	2348	2400	14740
30	2872	2035	10142	2418	2266	12432
31	3348	3033	15682	1455	1018	5317
32	2473	1865	9191	1594	1571	8945
33	2320	1725	9312	2772	2606	14442
34	3011	2349	12183	3363	3349	19579
35	3084	2232	11878	1738	1702	9919
36	3475	2595	13938	964	638	3926
37	2599	1595	8182	1892	1793	9870
38	3346	3145	17712	2723	2692	14505
39	3296	2572	12990	569	131	724
40	1157	0	0	3628	3686	20355

Table B-2 (Cont): Timber removals by stand under GA plan.

	Total Removals			***Timber remaining in 2046***		
Stand	merch cu ft	sawlg cu ft	sawlg bd ft	merch cu ft	sawlg cu ft	sawlg bd ft
41	4236	3471	18469	358	156	761
42	3132	2528	13261	1366	1058	5830
43	4587	3796	20996	156	0	0
44	2635	578	2942	1916	1195	6892
45	3076	2273	11860	1157	731	4468
46	2174	1699	8611	2355	2286	12315
47	4093	3645	22393	840	541	3131
48	5447	5259	28185	324	309	1701
49	5591	5606	30487	1882	1516	9477
50	2806	2107	10746	1489	1252	7266
51	3618	2788	14207	533	234	1268
52	1811	1371	7158	2252	2152	12095
53	3848	3426	20074	1454	1427	8856
54	3830	3367	18771	372	322	1893
55	3371	2976	15137	919	746	3971
56	2783	2419	12314	2110	1929	10700
57	2424	1631	7878	3436	3464	20426
58	4924	205	966	1264	10	52
59	3919	3317	17227	519	452	2558
60	3723	3694	18828	5192	5347	31211
61	2378	1636	8467	2461	2106	12324
62	4438	3582	18083	242	0	0
63	7320	6992	53197	2993	2577	17001
64	4360	3369	17665	229	183	982
All Stands	**431317**	**334984**	**1852952**	**208348**	**174705**	**1015554**