# AN ABSTRACT OF THE THESIS OF

Heidi L. Wedin for the degree of Master of Science in Forest Resources presented on March 12, 1999. Title: Stand Level Prescription Generation under Multiple Objectives.

Abstract approved:_ Signature redacted for privacy.

K. Norman Johnson

A stand level model, PREscription generator under Multiple Objectives (PREMO) was built to generate prescriptions that address multiple objectives for management of the forests of the Applegate River Watershed. PREMO is a part of the landscape model of the Applegate River Watershed Forest Simulation Project and generates prescription choices for the landscape simulation. Possible goals at the stand level, in addition to present net value, include limiting fire hazard, limiting insect hazard, enhancing structural complexity, maintaining snags and down wood for wildlife habitat, and enhancing fish habitat. PREMO finds good but not necessarily optimal solutions using the RLS-PATH algorithm (Yoshimoto, 1990) with a multi-stage look-ahead. The goal programming objective function maximizes the present net worth minus the squared sum of deviations from the forest structure goals. Placing scalar multipliers on desired goals and choosing target levels for goal measurements creates a goal emphasis. At each stage, PREMO chooses the prescription with the highest PNV from among the prescriptions that maximize the objective function. Variables in PREMO are live trees, snags, and down woody debris by species and diameter class. Growth is simulated with relationships from the Forest Vegetation Simulator (Dixon et al., 1995). Several prescriptions were generated for a young pine stand using different goal emphases. Using PREMO to find prescriptions that come close to meeting all goals seems generally possible in this example, while at the same time producing a positive PNV. Meeting the fire behavior and effects targets proved to be the most difficult to meet in every period. A conflict when achieving multiple goals exists between creating large snags and maintaining large trees. A number of improvements in modeling could be made

including recognizing species composition as a goal, developing a more sophisticated way of considering snags, and considering regeneration within the RLS-PATH function.

# Stand Level Prescription Generation under Multiple Objectives

by

Heidi L. Wedin

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 12, 1999
Commencement June 1999

Master of Science thesis of Heidi L. Wedin presented on March 12,1999.

APPROVED:

Signature redacted for privacy.

Major Professor, representing Forest Resources

Signature redacted for privacy.

Chair of Department of Forest Resources

Signature redacted for privacy.

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Signature redacted for privacy.

Heidi L. Wedin, Author

## ACKNOWLEDGMENTS

Many people have been instrumental in the completion of this degree. I thank Dr. Norm Johnson, my major professor, for the many hours spent helping me with the structure of this project and help with the thesis. I thank Dr. John Sessions for his expert advice and patience with the modeling efforts. I thank Dr. Jim Agee and Dr. Chris Maguire for their contribution to the goal measurements and targets. I also thank Don and Ellen Goheen for their contribution to the insect caused tree mortality relationships.

This project was made possible through funding by an endowment to do policy work on forests of Oregon provided by Boise Cascade. It was completed in partnership with the Applegate Partnership, Siskiyou and Rogue River Districts of the Forest Service, and the BLM Medford District.

Finally, there were many outside of the project team who assisted me during the course of this project. I must thank my friends and colleagues of the Harris Lab who made the hours spent working there very interesting. I am grateful to Patty Duke, my mentor and friend, for her prayers, support, and wise council. I thank Mark Kincaid for his unwavering support, loving consolation, and encouragement. I thank my family for their unconditional love. Most of all, I thank the Lord Jesus for giving me grace to finish, and strength in patience, perseverance, and wisdom. To Him be the glory.

And He has said to me, "My grace is sufficient for you, for My power is perfected in weakness." Most gladly, therefore, I will rather boast about my weaknesses, that the power of Christ may dwell in me.

II Corinthians 12:9

# TABLE OF CONTENTS

Page

## TABLE OF CONTENTS (Continued)

## TABLE OF CONTENTS (Continued)

# LIST OF FIGURES

# LIST OF TABLES

## LIST OF APPENDIX FIGURES

## LIST OF APPENDIX TABLES

## LIST OF ACRONYMS

| | |
|---|---|
| AMA | Adaptive Management Area |
| ARWFSP | Applegate River Watershed Forest Simulation Project |
| BLM | Bureau of Land Management |
| CFI | Forest Inventory |
| CVS | Current Vegetation Survey |
| DWD | Down Woody Debris |
| FBI | Fire Behavior Index |
| FEI | Fire Effects Index |
| FVS | Forest Vegetation Simulator |
| HLC | Height to Live Crown |
| MCF | Thousand Cubic Feet |
| MS-PATH | Multi Stage Projection Alternative Technique |
| PAG | Plant Association Group |
| PATH | Projection Alternative Technique |
| PNV | Present Net Value |
| PREMO | Prescription generator under Multiple Objectives |
| PRIME | Pacific Resource Inventory, Monitoring, and Evaluation Program |
| QMD | Quadratic Mean Diameter |
| RLS | Region Limited Strategy |
| TPA | Trees per Acre |

# Stand Level Prescription Generation under Multiple Objectives

# 1 INTRODUCTION

## 1.1 Introduction

This work is a contribution to the Applegate River Watershed Forest Simulation Project (ARWFSP). The goal of the ARWFSP is to develop a landscape simulation model for the Applegate River Watershed in southwest Oregon (1.1). If successful, the ARWFSP will provide a tool useful to the Applegate Partnership (a collaborative planning group in the Applegate Valley), land management agencies, and others in selecting polices and practices to achieve overall goals for the watershed. The landscape simulator will cover all forest land in the watershed, but choices for management will emphasize federal land.

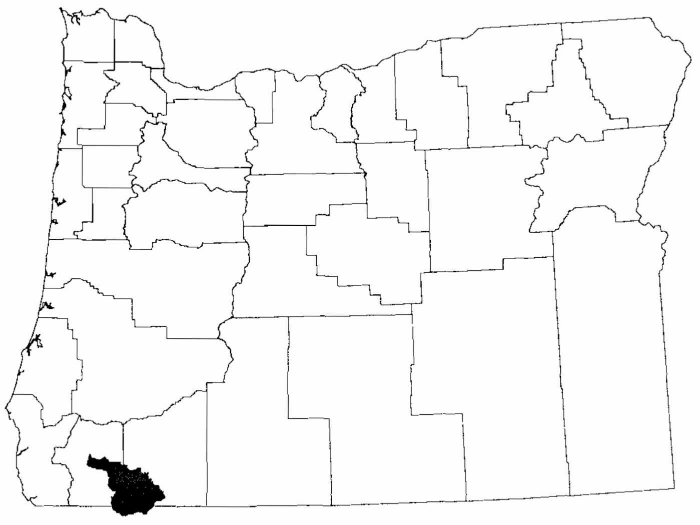


Figure 1-1. Applegate River Watershed in Oregon

## 1.2 The Applegate River Watershed

The project area is the Applegate River Watershed, comprised of approximately 500,000 acres. It is located in southern Oregon (1-1), west of Medford and south of Grants Pass. Federal land (Forest Service and Bureau of Land Management [BLM]) covers about two-thirds of the watershed and contains almost 80 percent of the forested land. Federal land in the watershed is designated an Adaptive Management Area (AMA) (USDA Forest Service and USDI Bureau of Land Management, 1994). Private non-industrial land occupies much of the remainder of the watershed, especially along streams and river bottoms.

## 1.3 Current Conditions and Management Guidelines

According to the Applegate AMA Ecosystem Health Assessment (USDI Bureau of Land Management, Medford District, USDA Forest Service, Rogue River National Forest, USDA Forest Service, Siskiyou National Forest, USDA Forest Service, PNW Research Station, 1994), the hazard of insect attacks is high across the entire watershed, with many pockets of extreme risk. Seral stage and stand structure has changed dramatically since pre-European settlement, resulting in younger, denser stands across the watershed. In these areas the stocking now exceeds the carrying capacity of the site. In addition, Douglas-fir and white fir now occupy areas that were traditionally more open-grown pine stands. These factors add to an increased danger of bark beetle outbreaks, as well as attacks from western pine beetle, mountain pine beetle, and pine engraver.

In the Applegate AMA Ecosystem Health Assessment (USDI Bureau of Land Management, Medford District, USDA Forest Service, Rogue River National Forest, USDA Forest Service, Siskiyou National Forest, USDA Forest Service, PNW Research Station, 1994), the BLM and the Forest Service created a joint report in which management recommendations and goals are set forth at the stand and at the landscape level. The general guidelines are to:

1. Reduce the density of trees (merchantable and non-merchantable) and shrubs by thinning and/or prescribed fire.
2. Protect and restore riparian areas and late successional habitat.
3. Increase the number of larger, older trees.
4. Promote, maintain, and restore shade intolerant species (such as ponderosa pine) in designated Plant Association Groups (PAG's).

The following suggested management strategies incorporate and build upon these guidelines.

1. First, in thinnings focus, on the residual stand by leaving large trees and fire resistant species, maintain a hardwood component, conserve and provide for future recruitment of snags and down woody material, protect aquatic resources, open the areas around pine trees, and leave non-host species in areas of root disease and dwarf mistletoe infection.
2. Second, the management prescription should minimize disturbance, reduce fuel ladders and ground fuels, treat slash, and develop a treatment plan that maintains low basal areas to reduce the risk of high levels of insect activity.
3. Finally, in light of the generally high level of fire and insect risk, treat as many stands as possible.

The Standards and Guidelines for Management Of Habitat For Late-Successional And Old-Growth Forest Related Species Within The Range Of The Northern Spotted Owl (USDA Forest Service and USDI Bureau of Land Management, 1994) also provides land management guidelines. "Adaptive Management Areas were selected to provide opportunities for innovation, to provide examples in major physiographic provinces, and to provide a range of technical challenges, from an emphasis on restoration of late-successional forest conditions and riparian zones to integration of commercial timber harvest with ecological objectives" (USDA Forest Service and USDI Bureau of Land Management, 1994, p. D-2).

## 1.4 Problem Definition

The goal of the ARWFSP is to provide a tool useful to the Applegate Partnership, land managers, and others of the Applegate Watershed community for the purpose of guiding the selection of forest management polices and practices. The project should help achieve the goals of AMAs, as set forth in the Standards and Guidelines for Management Of Habitat For Late-Successional And Old-Growth Forest Related Species Within The Range Of The Northern Spotted Owl (USDA Forest Service and USDI Bureau of Land Management, 1994).

AMAs are "designated to encourage the development and testing of technical and social approaches to achieving desired ecological, economic, and other social objectives" (USDA Forest Service and USDI Bureau of Land Management, 1994, p. D-1). One of the key features of AMAs is to "provide for development and demonstration of monitoring protocols and new approaches to land management that integrate economic and ecological objectives based on credible development programs and watershed and landscape analysis" (USDA Forest Service and USDI Bureau of Land Management, 1994, p. D-2). The guiding technical objective of AMAs is "scientific and technical innovation and experimentation" (USDA Forest Service and USDI Bureau of Land Management, 1994, p. D-2). Among the more specific technical objectives is the "design and testing of effects of forest management activities at the landscape level" (USDA Forest Service and USDI Bureau of Land Management, 1994, p. D-4).

The landscape model developed in this project is a technical approach in which the impacts of different management approaches can be tested at the landscape level. It is innovative in that management decisions respond to natural disturbances, and one can address different ecological objectives at the same time. Simulations are guided by ecological objectives, while economic effects also are considered. The landscape model has spatial orientation, thus disturbances, such as fire, affect specifically located stands in the watershed. The scope of the project is at the landscape level, while management decisions are made on a stand by stand basis.

The landscape model contains four stages. In stage one, prescriptions, designed to achieve overall landscape goals, are assigned to each "stand" recognized on the landscape. In stage two, the landscape is projected forwards in time, and the prescriptions are implemented. The landscape model simulates periodic natural disturbances in stage three. As these disturbances occur, stage four is invoked whereby the prescription for the stand is adjusted based on post-disturbance structure and simulation goals.

This thesis develops the prescription generator, PREMO (PREscription generator under Multiple Objectives), which is used in stages one and four of the landscape model. In stage one, PREMO creates management regimes for all stands over the entire planning horizon. In stage four it revises the prescriptions of disturbed stands for the remaining periods post-disturbance. The prescription generator selects a stand represented by a list of live trees, dead trees, and down woody debris, organized by species and diameter class, and creates several prescriptions for the stand in response to emphases on goals that might be used to guide management of the stands. In effect, this process creates different types of stands with varying abilities to respond to the stresses of natural disturbance, provide different types of wildlife and fish habitat, and produce different kinds and levels of outputs and amenities. The landscape simulator then uses the different prescriptions as management choices for each stand to meet goals specified for the entire landscape.

# 2 STAND LEVEL PRESCRIPTION GENERATION IN FORESTRY

## 2.1 Generic Stand Problem

The stand level management problem is a reoccurring theme in forestry. The problem is to identify the best strategy to reach a target state or stand structure. Decisions must be made regarding activities that can be carried out in the forest, specifically, how to alter the stand, by identifying which trees to remove, how to maintain the stand in regards to growing stock and species composition, and in which time period to undertake actions. Two tools, dynamic programming and multi-objective programming, can be integrated to make these decisions and they will be further described.

## 2.2 Dynamic Programming in Forestry

Dynamic programming (Dykstra, 1984) is a generalized approach for making a sequence of inter-related decisions in order to maximize overall effectiveness. Richard E. Bellman originally developed this method in the 1950s. It is useful in forestry problems, as many decisions are time-oriented and they must be sequentially implemented over time. The PATH algorithm, an adaptation of dynamic programming, will be discussed.

### <u>2.2.1 Introduction</u>

Dynamic programming is useful when a problem involves few state descriptors, meaning there are few decision variables. However, when there are many decision variables, time availability and computational capacity limit the ability to use dynamic programming. Therefore, forest managers have developed short cuts, simplifications, and heuristics to solve problems with many decision variables.

Some of these short-cuts are to assign solutions to neighborhood storage locations (Brodie and Kao, 1979), to use a network approach by optimizing only to the next period (Paredes and Brodie, 1987), to optimize the number of look-ahead periods (Yoshimoto et al., 1988), to minimize infeasibilities (Yoshimoto et al., 1994), and to use region limited strategy at a given stage to determine the optimal thinning levels of each of the diameter classes (Yoshimoto et al., 1994).

## 2.2.2 Dynamic Programming Components

Dynamic programming is well suited to forestry problems, as decisions are made at intervals in time that are easily represented by stages. State variables are the measurements for the possible states of a stand at a given stage, reflecting that a stand at one stage can have unique structural characteristics depending on the actions taken previously. The state variables also contain all information necessary for making decisions. Decision variables are the components of the management action and are subject to change in order to find the optimal level. In the forest management example, states usually represent a measure of the stocking and the species composition of the stand. Sometimes the state variable is not the same as the decision variable, but the decision variable is used to determine the value of the state variable. For instance, Kao & Brodie (1979) used a three-state descriptor model in which one state variable was basal area. The decision variables were trees per acre and type of thinning, but the combinations of number of trees removed and type of thinning produced the resulting basal area of the stand.

The number of conditions of a state variable depends on the node interval, the increment of precision of the decision variable. For example, if the state variable is trees per acre, then the node interval could be 20 trees, so that in the thinning decision you could thin to 100, 80, 60… trees per acre.

In the solution process, “dynamic programming starts by considering only one stage of the problem and finds the optimal solution for this stage. It then adds a second stage, finding a new optimal solution from the previous one, and so on until the problem is solved in its entirety" (Dykstra, 1984, p.291). The algorithm finds the optimum by using a recursive relation. For a backward recursion, the optimal solution is found for every state at a stage j, given that the optimal solution for each state at stage j+1 is known. A forward recursion is similar, but the optimal is found given that the stage j-1 is known. A backward recursion only yields the solution for one final stage. By beginning the optimizing procedure from the beginning stage (forward recursion), the optimal path to every stage of the network is known (Dykstra, 1984).

## 2.2.3 Dynamic Programming and the Two-State Variable Problem

Brodie et al. (1978) developed a forward-recursion dynamic programming algorithm to look at the impacts of adding variables reflecting various economic and growth potential changes on thinnings and rotation schedules. The decision variable was volume and the state variables were volume and age. This is a very simple model because volume is solely a function of age. The objective function was to maximize present net worth, which is a function of revenue from thinnings and final harvest minus logging costs. The economic and growth potential changes that they investigated were interest rates, regeneration costs, site potential, quality premiums and variable logging costs. Increasing the interest rate resulted in shorter rotation with earlier heavier thinnings. As regeneration costs increased, rotation length increased as well. Regimes with higher entry costs had longer thinning entries with heavier thinnings. Stands on higher site class lands had longer rotations. Since the stand was only represented by volume and age, quality premiums had to be a function of age. Quality premiums are increased revenues received for logs with larger diameters. However, quality premiums are also a function of diameter which is affected by stand density and, therefore, by thinning intensities. The added benefit in diameter premiums caused by thinnings was not realized in this model.

Brodie et al.'s (1978) model is guaranteed to find the optimal solution. In addition, it can be used to investigate changes in thinning regimes and rotations based on varying other factors in the problem. The negative side is that it is very simplistic and does not consider differences in volume per acre of the stand, thinning strategies or stand density.

### 2.2.4 An Algorithm with Three State Variables

Brodie and Kao (1979) developed another forward recursion dynamic programming algorithm with three state variables: number of trees, basal area of the stand, and stand age. The decision variable was trees per acre. The Quadratic Mean Diameter (QMD) of the stand was determined from number of trees and basal area. The benefit of this model is that with the addition of the QMD of the stand, increases in diameter due to thinnings can affect the optimization. However, the addition of a third state descriptor increases the computational burden of the algorithm dramatically. Brodie and Kao's (1979) solution to this problem was to set up neighborhood storage locations. The boundaries of the storage locations were set at intervals of merchantable trees and merchantable basal area to form a grid. As a result, there could be a range of solutions in one square of the grid, however, for any path that reached a given storage location, only the one with the highest present net worth was stored.

The potential problem with neighborhood storage locations is that a path which maximizes present net worth might not be the maximum in its storage location at every stage, and therefore the maximum could be missed. The benefit is that this method can efficiently "represent the continuous production-surface with a limited number of nodes" (Brodie and Kao, 1979, p.667).

### 2.2.5 Optimization with Different Thinning Types

Haight et al. (1985) introduced another state variable into the dynamic programming algorithm, creating a four-descriptor dynamic programming algorithm. The four state variables were stand age, residual number of trees, residual basal area, and thinning type. The decision variables were number of trees and thinning type. At each stage there were four possible types of thinning:

thinning from above, thinning from below, mechanical (proportional) thinning, and no thinning. The type of thinning determines what proportion of the trees of each diameter class are removed. For instance, in a thinning from below, a greater proportion of the trees harvested will be from the smaller diameter classes, so that the QMD of the stand after thinning will be higher than before the thinning. With a mechanical thinning, the final QMD will be the same, and with a thinning from above it will be lower. For a given number of trees harvested, the amount of basal area would be different depending on the thinning type. At every stage all four thinning types were projected and in each storage location the highest present net worth was stored including which type of thinning resulted in that value. With the addition of thinning type to the state variables, the algorithm can be used to compare many different combinations of numbers of trees in each diameter class that can be represented in the stand. This is useful when the management goal is to have a widely spaced stand or to create large diameter trees quickly.

Haight et al. (1985) further reduced storage requirements by classifying the tree list (plot-level per acre data for the stand) into 1-inch diameter classes. They used the average of the trees in each diameter class (the average height, tree diameter, and number of trees) to make a new list for the simulation. This reduces the computational efforts, however it also reduces the variability of the stand and does not reflect the true stand as accurately.

Haight et al. (1985) then used this model to determine harvest regimes guided by vigor indices that would reduce the susceptibility of stands to insect attacks. Goals of stocking control and tree diameters influenced the management regimes and resulted in harvest schedules that had a high vigor index and the desired range of diameters for the residual trees. They also looked at the effect of market premiums for large diameter trees and found that this scenario produced a thinning at age 30 that removed many small trees that actually had a negative present net worth. However, increased diameter growth made up for this loss because of the premium received for trees with larger diameters.

## 2.2.6 PATH Algorithm

Paredes and Brodie (1987) developed the Projection Alternative TecHnique (PATH) algorithm. In dynamic programming the value of the standing trees is not realized until the final stage, however the PATH algorithm maximizes the return from the thinning plus the value of the standing trees at the next stage at every node. The node that has the largest value becomes a part of the optimal path and the algorithm moves to the next stage to determine the optimal thinning level for that stage. Therefore the algorithm does not maximize over the entire planning horizon but only over the next period.

The PATH algorithm seeks to reduce the storage space required to arrive at an optimal solution. However, if the full effect of thinning is not captured by the next stage, the final solution is not guaranteed to be the optimal one. In this case a good solution is found which is not necessarily the optimal one. Two occasions where this occurs are described in 2.2.7 MS-PATH. The benefit of using the PATH algorithm is reduced computation time.

The PATH algorithm was compared to the three-state descriptor dynamic programming algorithm from Brodie and Kao (1979) with neighborhood storage locations. Paredes and Brodie (1987) presented two examples and compared the solution times for these using the PATH algorithm and dynamic programming and found that on a mainframe computer, the time savings ratio of using the PATH algorithm was around 30 times. However, there were some differences in the thinning regimes that could be a result of the neighborhood storage locations.

## 2.2.7 MS-PATH

Multi Stage Projection Alternative TecHnique (MS-PATH) (Yoshimoto et al., 1988) is an option to be used when a look-ahead of one period is not sufficient to capture the effect of a decision. This can occur when there is no thinning at the next stage based on a one period look-ahead or if the increased growth that results from the thinning is not realized within the period. The MS-PATH first determines the optimal look-ahead period at each stage then carries on with the regular PATH algorithm.

In an example, the authors only found a 2% increase in the objective function when using MS-PATH, which might not be worth the increased computational time. However, the MS-PATH algorithm produced different thinning regimes with fewer heavier thinnings.

## 2.2.8 RLS-PATH

Yoshimoto et al. (1990) used the PATH algorithm to look at a more precise break-down of the stand description into smaller diameter classes and species groups. Optimizing each of the diameter/species classes instead of harvesting a pre-specified proportion of the trees of the diameter classes of all species combined would produce more precise solutions and could improve the optimal regime. Since the computational burden with such precision was too great, even while using the one-period look-ahead, the authors incorporated the Region Limited Strategy (RLS). This strategy has two parts and seeks to produce a good solution at every stage of the problem. In the first part, each diameter class is optimized consecutively. Given a decision vector $T = (t_1, t_2,\ldots,t_n)$ where $t_i$ represents the number of trees harvested in the ith diameter class at the current stage, one variable $t_i$ is allowed to vary at a time. The variable that maximizes the objective function is fixed. With this variable fixed at the level that maximizes the objective function, the other variables are again allowed to change one at a time. A second diameter class with the greatest improvement in the objective function is now fixed. The second part of the RLS is to adjust the fixed thinning levels, given that a new diameter class has been fixed. This is a recursive process that runs until all fixed diameter classes reach a steady-state. It allows the first diameter class that was fixed to vary, given that a certain number of trees for the second class will be harvested. Then the first part begins again, finding a new diameter class to be fixed. This process is repeated until harvesting trees from one more diameter class does not improve the solution.

By performing a guided search over the solution space at a given stage, RLS reduces the storage space needed, however, not all possible solutions are evaluated. The difficulty caused by not looking at all possible combinations is that the optimal thinning levels might not be found, but the authors hope to find them by maximizing the return from thinning and looking ahead at the return from growth.

## 2.3 Multiple Objective Scenarios

### 2.3.1 Goal Programming

Goal programming (Dykstra, 1984) is a technique for solving multi-objective problems. It was developed by Charnes and Cooper in the early 1960s. Goal programming is an extension of linear programming, with a modified objective function and goal statement constraints. The objective function minimizes deviations from multiple goals. The goal constraints are different from regular constraints, in that the goal constraint is satisfied as closely as possible, however, it is possible not to meet the goal completely. When this occurs, a deviation from the goal is calculated. There is a penalty for not meeting the goal constraint, and these penalties, or deviations, are minimized in the objective function. The goal constraints can also be prioritized with scalar multipliers applied to the deviations. Goal programming is often used in natural resource planning, as forest lands are often managed under multiple objectives.

### 2.3.2 Stand Projection Under Multiple Goals

Two recent studies have used multiple-objective programming in stand projection. In the Sierra Nevada Ecosystem Project (Cousar, 1996), a stand level multi-objective optimization model was developed with a set of silvicultural and ecological goals that guided the optimization. Activities of timber harvest, prescribed burning, fuelbreak creation, and no activity were used to create a stand structure that most closely met the goals, such as increasing late successional structure and/or minimizing fire hazard. There were several simulations where emphasis was shifted among the silvicultural and ecological goals. The optimization process was performed with the PATH Algorithm and achievement of goals were evaluated from 10 to 30 years in the future. The goal programming objective function minimized deviations from the goals.

Haight et al. (1992) generated stand prescriptions for Rocky Mountain conifer stands using the Hooke-Jeeves method. The goal was to meet stand density targets that represented the visual quality and wildlife habitat of the stand. In this project, the prescription was made for one species group with three unmerchantable diameter classes and five merchantable diameter classes. The objective function, which drove the prescription generation, was maximization of revenue (or of volume production) minus the sum of the infeasibilities. Infeasibilities were measured as a squared deviation from the stand density target. A scalar multiplier was applied to the sum of the infeasibilities. In simulations, the authors set the scalar multiplier equal to "0" to find the unconstrained optimum, and then they set it to "1" to place equal weights on maximizing revenue or volume production and meeting stand density targets. The results showed that stand density targets in visually sensitive areas were attainable, while high stand densities for thermal cover in the wildlife simulations resulted in infeasible solutions.

## 2.4 Conclusion

Goal programming appears to be an appropriate framework for expressing the multiple objectives of the Applegate project. The quadratic penalty function used by Cousar et al. (1996) and Haight et al. (1992) is an intuitively appealing way of incorporating increasing marginal costs with increasing deviations from goals. Anticipating a likely problem size for 10 species and 50 diameter classes, over 20 periods, the computational efficiency of the RLS-PATH algorithm makes it attractive over traditional dynamic programming. Previous applications of RLS-PATH algorithm support this approach. Based on the literature reviewed, it appears that the use of goal programming with the RLS-PATH algorithm can be used to generate efficient prescriptions to meet ecological goals.

# 3 DESCRIPTION OF STAND STRUCTURE, GROWTH, AND MORTALITY

Each stand is characterized by a vegetation class. This describes the overstory forest structure and dead material.

## 3.1 Vegetation Classes

The Applegate landscape was categorized into 99 potential vegetation classes, (Table 3-1). These are defined as having the same dominant canopy, quadratic mean diameter (QMD) of the stand, and total canopy closure. The dominant canopy is determined by the percent crown canopy by species, resulting in the following groups:

1. Red fir
2. Mixed conifer with elevation greater than 3000 feet
3. Mixed conifer with elevation less than 3000 feet
4. White fir
5. Pine
6. Closed cone pine
7. Deciduous hardwood
8. Conifer/Hardwood
9. Evergreen hardwood (evergreen hardwood species)

As time passes, activities, tree growth, and tree mortality from insects, fire, strong wind events, and root disease change the character of the stand. As this occurs, stands will move through the matrix of vegetation classes. The initial acres in each vegetation class based on the interpretation of satellite imagery, are displayed in Table 3-1. A tree list was created for each vegetation class by classifying plot data from forest surveys (Forest Service Current Vegetation Survey (CVS), Pacific Resource Inventory, Monitoring, and Evaluation program (PRIME), Bureau of Land Management CVS and Forest Inventory (CFI)).

| Quad Mean Diam range | % Crown Clos-ure | Red fir | Mixed Conifer <3000’ | Mixed Conifer >3000’ | White fir | Pine | Closed Cone Pine | Decidu-ous Hard-wood | Conifer/ Hard-wood | Ever-green Hard-wood | Total: | % of Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 – 4.9 | < 60 | 572.6 | 1085.9 | 1662.6 | | 1951.6 | 904.0 | 4568.6 | 428.6 | 7806.1 | 18980 | 4.61 |
| 5 – 8.9 | < 60 | | 1074.5 | 3825.1 | | 3709.5 | 372.5 | 5373.3 | 18561.4 | 1587.9 | 34504.2 | 8.38 |
| | > 60 | | | | | 437.8 | 368.8 | | | | 806.6 | 0.20 |
| 9 – 14.9 | < 60 | | 779.9 | 350.9 | 1298.6 | 10352.9 | | 20492.7 | 14253.8 | 883.0 | 48411.8 | 11.75 |
| | > 60 | | 8854.5 | 8009.1 | | 15054.5 | | 7122.6 | 50876.0 | 17772.1 | 107688.8 | 26.14 |
| 15 – 20.9 | < 60 | 1714.3 | 13.0 | 1797.2 | 1070.1 | 4354.6 | | 3685.2 | 2416.1 | 71.6 | 15122.1 | 3.67 |
| | > 60 | 3196.0 | 30346.4 | 27909.6 | 4732.8 | | | 3401.2 | 39802.4 | 5523.5 | 114911.9 | 27.90 |
| 21 – 24.9 | < 60 | 848.6 | 660.9 | 643.3 | 2592.7 | 928.4 | | | | | 5673.9 | 1.38 |
| | > 60 | 3199.1 | 319.4 | 3868.5 | 4776.2 | | | | 310.9 | | 12474.1 | 3.03 |
| 25 – 31.9 | 0-100 | 1936.0 | 4022.5 | 27518.1 | 12492.5 | 1442.0 | | | | | 47411.1 | 11.51 |
| 32+ | 0-100 | | 21.8 | 5570.6 | 342.2 | | | | | | 5934.6 | 1.44 |
| Total: | | 11466.6 | 47178.8 | 81155 | 27305.1 | 38231.3 | 1645.3 | 44643.6 | 126649.2 | 33644.2 | 411919.1 | |
| % of | Total: | 2.78 | 11.45 | 19.70 | 6.63 | 9.28 | 0.40 | 10.84 | 30.75 | 8.17 | | 100.0 |

Table 3-1. Acres in Vegetation Classes

This table displays forested acres in the Applegate River Watershed in 1993.

The watershed is also subdivided into Plant Association Groups (PAGs). Geology, elevation, slope, aspect, and/or precipitation define a PAG. PAGs are used to identify the site quality of each part of the watershed, and also to help target levels for the measurement of stand goals (see 4.2.2 Limit Insect Hazard).

In the Applegate River Watershed the PAGs are:

1. Jeffrey pine
2. Pine/oak
3. Douglas-fir/dry
4. Douglas-fir/wet
5. White fir/dry
6. White fir/wet
7. Red fir

In the results section of this thesis, several optional prescriptions are demonstrated for a pine stand, QMD 9-15, canopy closure > 60, that is in the pine/oak PAG.

## 3.2 Growth Simulation

In order to project tree growth, the routines for diameter and height growth, change in crown ratio, and crown width calculations for nine species were taken from the Forest Vegetation Simulator (FVS), Northern California/Klamath Mountains Variant. FVS is written in FORTRAN. For the purpose of decreasing computational time and adjusting mortality functions, these subroutines were translated into the "C" programming language and directly incorporated into PREMO. The results of these equations are close to those of the FVS subroutines.

### 3.2.1 Comparison of diameter growth in FVS and PREMO

The following tables compare the diameter growth between FVS and PREMO for each species over 100 years for trees with an initial DBH of 10", 16", 18", and 24". The percentage difference ranged from 0.02% to 6.46%. The average percent difference across all four tests was 2.37%. PREMO generally underestimates diameter-growth for smaller trees, and overestimates for larger trees.

| Species | FVS | PREMO | Percent Difference |
|---|---|---|---|
| Black Oak | 15.8 | 15.53 | -1.71 |
| Douglas-Fir | 32.7 | 32.15 | -1.69 |
| Incense Cedar | 26.7 | 26.02 | -2.55 |
| Pacific Madrone | 27.6 | 27.15 | -1.62 |
| Ponderosa Pine | 30.6 | 30.18 | -1.37 |
| Red Fir | 29.8 | 28.68 | -3.77 |
| Sugar Pine | 29.5 | 28.57 | -3.16 |
| Tanoak | 23.3 | 22.78 | -2.22 |
| White Fir | 26.1 | 25.41 | -2.66 |

Table 3-2. Comparison of Diameter Growth of a 10" DBH tree between FVS and PREMO.

| Species | FVS | PREMO | Percent Difference |
|---|---|---|---|
| Black Oak | 21.9 | 21.69 | -0.95 |
| Douglas-Fir | 35.4 | 34.82 | -1.64 |
| Incense Cedar | 30.9 | 31.10 | 0.65 |
| Pacific Madrone | 33.2 | 32.8 | -1.20 |
| Ponderosa Pine | 32.9 | 32.43 | -1.43 |
| Red Fir | 34.0 | 33.15 | -2.50 |
| Sugar Pine | 32.0 | 31.46 | -1.69 |
| Tanoak | 29.6 | 29.09 | -1.72 |
| White Fir | 30.4 | 29.64 | -2.51 |

Table 3-3. Comparison of Diameter Growth of a 16" DBH tree between FVS and PREMO.

| Species | FVS | PREMO | Percent Difference |
|---|---|---|---|
| Black Oak | 25 | 25.05 | 0.21 |
| Douglas-Fir | 43.7 | 46.37 | 6.11 |
| Incense Cedar | 38.4 | 39.62 | 3.19 |
| Pacific Madrone | 36.8 | 36.69 | -0.31 |
| Ponderosa Pine | 39.5 | 41.75 | 5.70 |
| Red Fir | 47.5 | 48.10 | 1.25 |
| Sugar Pine | 50.6 | 51.53 | 1.84 |
| Tanoak | 32.9 | 33.08 | 0.56 |
| White Fir | 34.5 | 36.73 | 6.46 |

Table 3-4. Comparison of Diameter Growth of an 18" DBH tree between FVS and PREMO.

| Species | FVS | PREMO | Percent Difference |
|---|---|---|---|
| Black Oak | 30.7 | 30.72 | 0.07 |
| Douglas-Fir | 46.9 | 46.90 | 5.35 |
| Incense Cedar | 43.5 | 43.95 | 1.04 |
| Pacific Madrone | 40.2 | 40.00 | -0.49 |
| Ponderosa Pine | 42.0 | 44.03 | 4.83 |
| Red Fir | 49.3 | 51.79 | 5.05 |
| Sugar Pine | 52.9 | 54.71 | 3.42 |
| Tanoak | 37.9 | 37.91 | 0.02 |
| White Fir | 39.2 | 40.93 | 4.40 |

Table 3-5. Comparison of Diameter Growth of a 24" DBH tree between FVS and PREMO.

### <u>3.2.2 Comparison of Height and Crown Ratio</u>

Due to the difference in mortality functions of FVS and PREMO, direct comparison of height growth and change in crown ratio is difficult. The following table gives a rough comparison for a ponderosa pine stand after 100 years of growth. The percent difference in PREMO from FVS for DBH ranges from 0.28% to 3.69%, height ranges from 0.90% to 8.83%, and crown ratio ranges from 2.56% to 50%.

| FVS | | | | PREMO | | | |
|---|---|---|---|---|---|---|---|
| TPA | DBH | HT | CR | TPA | DBH | HT | CR |
| 4.90 | 34.3 | 126.8 | 17 | 2.82 | 35.45 | 115.6 | 21 |
| 7.60 | 39.4 | 126.9 | 12 | 6.57 | 40.33 | 116.3 | 13 |
| 10.16 | 42.4 | 126.9 | 10 | 8.75 | 42.90 | 120.6 | 13 |
| 4.19 | 43.6 | 126.9 | 10 | 3.65 | 45.21 | 125.3 | 15 |
| 9.80 | 50.1 | 126.9 | 38 | 8.40 | 49.43 | 115.7 | 36 |
| 2.30 | 54.4 | 126.9 | 39 | 1.93 | 54.25 | 115.7 | 40 |
| 1.80 | 56.6 | 144.0 | 15 | 1.58 | 56.03 | 145.3 | 17 |

Table 3-6. Comparison of Height and Crown Ratio

### 3.2.3 Growth Sequence

Growth is calculated for one period at a time, which is equal to five years. The order in which growth is calculated is depicted in Figure 3-1. The code for the programs for diameter growth can be found in Appendix A.

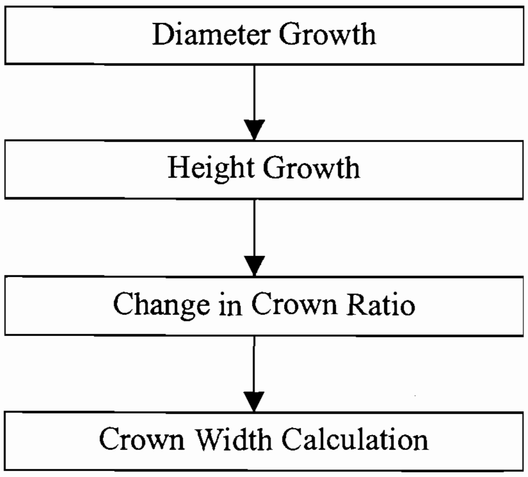


Figure 3-1. Modules Used for Calculating Growth in FVS

## 3.2.4 Regeneration

A natural regeneration model was developed by Kayser et al. (1998) which is invoked every other period to introduce in-growth into the tree list. The tree list of the regeneration is not introduced until 10 years after the disturbance occurred. This is because of the poor ability of the model to handle trees without a DBH. The species mix of the regeneration is dependent on the before-treatment stand, with tolerant species responding in greater numbers under greater residual stand basal areas. The number of trees added is given by Equation 3-1 (Kayser et al. 1998). The diameters for the regenerated trees were taken from the plot data for this project, while the heights were calculated using FVS.

$$\textit{TPA added} = 625 * e^{(-0.02\ *\ \textit{stand basal area [square feet per acre]})}$$

Equation 3-1.

## 3.3 Mortality Functions

The ARWFSP scientists do not feel that empirical growth and yield models, such as FVS, are able to fully represent the episodic nature of mortality in the Applegate River Watershed, especially in large trees. Therefore, they developed mortality functions to simulate tree death. These are organically driven equations, meaning that tree death results from a natural disturbance. As a result, the prescription generator does not have direct relationships between tree mortality and suppression. The project scientists categorized mortality functions into 2 groups, depending on whether mortality events are periodic or episodic in nature.

In the ARWFSP model, the mortality in the landscape is assumed to result from fire, strong wind events, insects, and root disease in high elevation stands. The prescription generator reflects the periodic (endemic) levels of mortality for insects, wind and root diseases. The occurrence of mortality due to episodic natural disturbance (fire, insect, and wind) events is applied at the landscape level, not in PREMO, due to the spatial and probabilistic nature of the disturbances.

### 3.3.1 Tree Mortality Related to Insects

ARWFSP scientists split mortality resulting from insects into two categories, periodic and episodic. They felt that scattered mortality resulting from insects occurs in every period, regardless of the amount of precipitation. However, episodic mortality is drought driven and occurs in addition to periodic mortality. The episodic insect hazard is modeled at the landscape level, not in PREMO.

PREMO simulates periodic mortality in every period. The mortality relationships, described in Agee and Goheen (1998), give the mortality as 0.05% of the stand basal area every period. Mortality trees are selected in descending order of tree diameter. However, the insects kill not only large trees, but smaller trees as well. Therefore, if the stand has trees greater than 20 inches in diameter, the mortality comes from two groups, the largest trees in the stand, and trees smaller than 20 inches in diameter. The former incurs 67% of the mortality if possible, with the rest coming from trees with a diameter less than 20 inches and ending with 8 inches. These insects do not generally kill trees with a DBH less than 8 inches.

## 3.3.2 Wind Disturbance

Project scientists split mortality resulting from wind events into two categories, periodic and episodic. Wind disturbance, described in Agee and Goheen (1998), is significant in stands that are high elevation, well-stocked, and tall. The wind disturbance team projected that wind events in the Applegate River Watershed are limited to stands that meet the following conditions: stand elevation is above 4000 feet, stand basal area per acre is greater than 120 square feet, the tallest five trees per acre are greater than 50 feet, and the aspect is from 45° to 315°.

Table 3-7 describes the probabilities and results of wind disturbance in four wind strength categories. Only light wind disturbance is modeled in PREMO, the others will be modeled in the landscape simulator.

| Wind Category | Frequency (years) | Yearly Probability | Loss of Stand (percent of stand basal area) |
|---|---|---|---|
| Light (periodic) | 1 | 1 | 0.25 |
| Moderate | 25 | 0.04 | 1.0 |
| Severe | 50 | 0.02 | 5.0 |
| Catastrophic | 200 | 0.005 | 10 |

Table 3-7. Wind Disturbance

PREMO simulates periodic mortality in every period. Trees succumbing to windthrow are selected in descending height order until 0.25% of the stand basal area is killed (Agee and Goheen, 1998).

### 3.3.3 Douglas-Fir Dwarf Mistletoe and Root Diseases

According to Agee and Goheen (1998), we lack data to quantify the percentage of stands infected by the Douglas-fir dwarf mistletoe and the severity of it in the Applegate Watershed. However, we believe that the mortality related to Douglas-fir dwarf mistletoe will be accounted for in the Douglas-fir bark beetle mortality predictions (see 4.2.2. Limit Insect Hazard).

The insect and disease natural disturbance group determined that tree mortality as a result of root diseases would be modeled entirely as periodic natural disturbance. Therefore, PREMO will simulate all of the root disease mortality in this project; none will occur in the landscape simulator.

Mortality caused by root diseases in the lower elevations will be encompassed by bark beetle mortality. However, it is assumed that at high elevations insect mortality will not sufficiently portray mortality due to root diseases. The following modeling guidelines were obtained from Agee and Goheen (1998) for mortality in white fir/wet and red fir PAG's which correspond to higher elevations in the watershed.

In the white fir/wet and red fir PAG's, root disease centers comprise approximately 8% of the area. In a 100-year period the expected value for loss of volume of susceptible species is 50%. The tree species in order of susceptibility to root disease are white fir, red fir, then Douglas-fir. Stands are marked permanently for having root disease centers, however, non-host species can be grown in these areas. In the prescription generation 2.5% of the basal area is killed in each period. Mortality is spread evenly among all susceptible species (Agee and Goheen, 1998).

# 4 QUANTIFYING GOALS AND MEASURES FOR THEIR ATTAINMENT

## 4.1 Introduction

ARWFSP scientists developed potential stand goals by reviewing key issues of interest to the people of the Applegate River Watershed and in working with employees of agencies who manage land in the Watershed. Fire and insect disturbances are influenced by drought periods, therefore the landscape simulation will model weather streams, which will influence these disturbances.

## 4.2 Potential Goals for the Stands

The ARWFSP wishes to include a number of goals for the management of forests in the landscape simulator. These goals are to achieve a landscape condition that has a low fire and insect hazard, enhances fish and wildlife habitat, and provides a positive PNV. The landscape is composed of stands of different initial condition. Potential goals for individual stands are listed in Table 4-1 in a non-ranked order. As can be seen, goals are based on different stand attributes.

| Goal | Based on | Measures of goal attainment |
|---|---|---|
| Limit fire hazard | Stand condition: structure | Flame length, potential for torching, crown bulk density, percent of basal area that would burn in a 90% fire |
| Limit insect hazard | Stand condition: stocking | Basal area thresholds |
| Enhance wildlife habitat | Stand condition: structure | Vertical structure, snag number and size, linear feet of Down Woody Debris (DWD) |
| Improve fish habitat | Stand condition: distribution and size | Number of large trees, canopy closure in riparian areas |
| Positive Present Net Value | Timber harvest | PNV |

Table 4-1. Stand Goals

Measurement of goals based on stand condition are calculated by investigating the residual tree list (standing live and dead trees, and down wood), while measurement of the goal based on stand output is calculated from the harvest tree list. Tools available for manipulating stand condition to achieve goals are growth, tree harvest, and snag creation.

### 4.2.1 Limit Fire Hazard

One stand level goal is to minimize the risk of severe fires. The method of decreasing fire hazard in PREMO is by growing or harvesting trees. There are four stand structural conditions that increase the risk for severe fire:

1. High fuel loadings that increase flame lengths.
2. Trees with small diameters, and therefore less resistant bark, which have higher probability of mortality for a given flame length.

3. Low effective Height to Live Crown (HLC) that allows fire to reach from the forest floor to the canopy of the dominant trees (fuel ladders).
4. High crown bulk densities which represent crowns that would carry the fire through the canopy if the fire were to reach it.

In PREMO, two fire indices are used to measure the risk of severe fire hazard. The Fire Behavior Index (FBI) (Equation 4-1. Fire Behavior Index) describes what kind of fire the stand will have and the Fire Effects Index (FEI) measures the result of the fire on the stand.

*Fire Behavior Index = 50/15*flame length(ft) +30*torching +20*absolute value of (crown bulk density (kg/m*$^3$*) -0.1) / 0.2).*

Equation 4-1. Fire Behavior Index

*Fire Effects Index = % basal area (sqft/acre) killed by 90% worst-case weather fire*

Equation 4-2. Fire Effects Index

The FBI is a combination of the potential flame length (ft) of the stand, torch potential, and the crown bulk density (kg/m$^3$) of the canopy. The fire behavior index can range from 0.33 to 100, where a higher number represents a more dangerous fire hazard. An initial flame length is assigned to a stand depending on its original vegetation class. Following that the flame length increases or decreases according to the action taken in that stand in that period. The current numbers are rough estimates of the actual changes and will be updated when the fire model is complete. Currently, if no trees are harvested, the flame length increases by 0.33 feet. If there is a harvest, the d/D ratio is calculated. The d/D ratio divides the average diameter of the pre-treatment stand by the average diameter of the harvest. If the d/D ratio is less than 0.9, this signifies a thinning from below and the flame length is decreased by 0.33 feet. Otherwise the flame length remains the same.

Torching occurs when a ground fire is carried into the canopy. The effective stand height to live crown (HLC) is the lowest crown height that would result in a crown fire. The fire disturbance group of the AWRFSP determined that the effective stand height to live crown would be determined in the following manner:

1. The HLC is determined for every tree on an acre in a given stand.
2. These are placed into height classes of 1m.
3. The height class of the 50th tree or the 1/10th tree, whichever is less, starting from the lowest HLC, determines the stand HLC.
4. Then the HLCs in the next three higher height classes are tallied.
5. If they exceed 5% of the TPA of the stand, or five trees, whichever is less, then the effective stand HLC is the stand HLC.
6. If not, the next higher height class is chosen as the stand HLC, and the program determines if there are sufficient trees in the next three classes to make that stand HLC the effective stand HLC.

This height is used to calculate a critical flame length, which is a combination of stand flame length and heat of ignition. If the simulated flame length is greater than the critical flame length, torching occurs, which could lead to a canopy fire if the crown bulk density is high enough. The crown bulk density measures if a canopy fire will occur given that torching occurs. The higher the crown bulk density, the higher chance in the stand of a canopy fire.

For an example of FBI (Equation 4-3), a stand with flame length of 0.33 feet, torching, and a crown bulk density of 0.15 kg/m$^3$, would have a fire index of 36.1. The target level for the FBI was set at 30. This means that a stand with structure that results in an FBI less than 30 has minimal fire risk. Any stand that has an FBI $\leq 30$ does not have a positive torch potential.

$$\textit{Fire index} = 50/15*(\mathbf{0.33}\,ft) + 30*(1) + 20*abs((\mathbf{0.15}\,kg/m^3 - 0.1)/0.2 = 36.1$$

Equation 4-3. Example of a Fire Behavior Index

The FEI Index (Equation 4-2) calculates the percent average basal area that would be killed in the stand. PREMO calculates the basal area that would be killed for each species, diameter-class and then averages them for the stand. Basal area that would burn depends on the flame length and diameter of the tree. The target level for the FEI was set at 30, to allow a stand of some magnitude to exist after harvest. As the target level was set lower and lower, the model would cut larger and larger trees, since even large trees have a percent of basal area that would burn in a fire. As an example, if the FEI is less than 10 for a Ponderosa pine stand, the average diameter of the stand after harvest would have to be greater than 28" to meet the target level (Table 4-2

Table 4-2 shows which trees to harvest in a simple stand where all trees have the same diameter. It describes the minimum tree DBH that would be left after harvest to meet a certain FEI target level.

| Species | Fire Effects Index (FEI) | Cut all trees less than this DBH (in) | |
|---|---|---|---|
| | | Flame Length 0-2 (ft) | Flame Length 2-4 (ft) |
| BO | 21 - 30 | 20 | 36 |
| | 11 - 20 | 30 | All |
| | 0 - 10 | All | All |
| DF | 21 - 30 | 14 | 14 |
| | 11 - 20 | 16 | 16 |
| | 0 - 10 | 26 | 26 |
| HW | 21 - 30 | All | All |
| | 11 - 20 | All | All |
| | 0 - 10 | All | All |
| PP | 21 - 30 | 12 | 12 |
| | 11 - 20 | 16 | 16 |
| | 0 - 10 | 28 | 28 |
| SP | 21 - 30 | 28 | 28 |
| | 11 - 20 | 36 | 36 |
| | 0 - 10 | All | All |
| WF | 21 - 30 | 18 | 18 |
| | 11 - 20 | 24 | 24 |
| | 0 - 10 | 38 | 38 |

Table 4-2. Minimum Average Tree DBH to be Left by Species and Flame Length, All Smaller Trees to be Cut, to Meet a Given Target Level for FEI.

Prescriptions can reduce the fire hazard of a stand by removing trees and treating fuel, to either reduce ground fuels, remove the ladder fuels, or reduce canopy closure to prevent crown fires from spreading in the stand. Such activities result in a reduced flame length and/or probability of fire spreading through the canopy should it crown.

## 4.2.2 Limit Insect Hazard

According to Goheen (1998) bark beetles and wood borers are a natural component of the Applegate River Watershed, but their roles have changed with fire exclusion. Severe mortality related to these insects is closely associated with basal area thresholds.

Project scientists estimate that the losses to insect mortality increase once a stand reaches a certain basal area (Agee and Goheen, 1998). These basal area thresholds are shown in Table 4-3. The goal will be to thin the stands to a level below the basal area threshold limit, so that the stand will not reach the limit before the next thinning. The threshold limit is set as the most limiting basal area threshold of the PAG, depending on the tree species composition of the stand.

| Plant Association | Target tree species | | |
|---|---|---|---|
| Group | Douglas-fir | White fir | Pines |
| Jeffrey pine | 80 | 0 | 80 |
| Pine/oak | 80 | 0 | 80 |
| Douglas-fir/dry | 120 | 0 | 120 |
| Douglas-fir/wet | 250 | 0 | 120 |
| White fir/dry | 250 | 120 | 180 |
| White fir/wet | No Threshold | 250 | 180 |
| Red fir | No Threshold | 250 | 180 |

Table 4-3. Basal Area Thresholds for Insect Hazard.

Tree mortality due to insects is classified into two levels: periodic and episodic. Episodic levels will not be modeled at the stand level for this thesis, however, goals to minimize such occurrences will. Losses to mortality due to episodic levels of insect activity can be managed by maintaining the stocking of stands below a basal area limit. In the landscape model, weather is simulated each period, with corresponding droughts in dry periods; insect mortality is simulated in drought years if a stand exceeds the basal area limits. The percentage of basal area of the stand that is killed increases and decreases with the drought intensity. For each insect, mortality only occurs in host-species.

### 4.2.3 Wildlife Habitat

The project uses three measurements of wildlife habitat quality: vertical structure, number and diameter of snags, and linear feet and diameter of DWD. Vertical structure measures the vertical distribution of crowns in the stands. The measure is derived by looking at height classes of five foot intervals, and the number of trees whose crowns intersect a given height class. Then the variance among the height classes is calculated. A low variance represents a stand with high complexity, while a high variance denotes a stand with a rather simple canopy structure.

The idea for measuring vertical structure is similar to that described in Dubrasich et al. (1997). Researchers calculated the crown area variance between heights for several structurally complex and simple forest stands in southwest Oregon and in the Willamette Valley. For this project, the standard deviation among number of crowns in height classes is used to determine vertical structure. Targets were set (standard deviation < 50 for complex stands, >100 for simple stands), by looking at crown distributions of differing variances. Examples of stands with complex (standard deviation = 14) and simple (standard deviation = 160) vertical structures are located in Appendix E.

The decay of snags and down wood was also modeled, using relationships from Mellen and Ager (1998). The initial snags and DWD per acre were taken from the plot database. Snags come from periodic mortality (insects, root disease, and wind events) or are created. Snags decay by period, in which some fall, and others lose height. Both of these are the source for DWD, as long as the condition of the snag was class I - IV. DWD decays in length and condition. The target for snags is five per acre greater than 16 inches in diameter. These target levels represent "natural" levels found by Ohmann et al. (1994).

Snag classes were originally defined by Cline et al. (1980). There are two fashions in which snags decay. One form is from top to bottom, resulting in decrease height and the loss of branches, needles, and bark. The other is wood deterioration from sapwood to heartwood, causing "softer" snags. This classification produces five classes, representing various degrees of deterioration of the snag, where class one a tree that died recently. In class five, all of the limbs and branches are gone, up to 20% of the bark is left and the sapwood is gone. This is the only type of snag that will not be counted in this project.

The target for DWD is 120 linear feet per acre. In order to be counted as DWD, the piece must have a large end diameter of at least 16 inches, a length of 16 inches and a DWD class of one through four. Maser et al. (1979) developed a five class classification system for DWD deterioration. All down logs are counted in this project, unless they are class five. In this case bark and twigs are absent, the texture is soft and powdery, and all of the log is in contact with the ground. DWD counted in the project (classes one through four), is less decayed. The DWD specifications were taken from the NW Forest Plan guidelines for matrix areas (USDA Forest Service and USDI Bureau of Land Management, 1994). The amount of dead, as measured in the prescription generator, will under-represent the actual amount of deadwood on the landscape, as more will result from episodic mortality, which is not present in the prescription generator.

### 4.2.4 Fish Habitat

In this project, fish habitat can be enhanced in two ways in stands that are in or near riparian areas. For stands located in riparian areas, one goal is to maintain high canopy closure. Stands located in or near riparian areas carry a goal of number of large trees per acre. These large trees would be present to create potential down wood in the body of water to improve fish habitat, should they fall and land in the water. The goal is tentatively to maintain five large trees per acre in the stand. A large tree is defined as having a DBH $\geq$ 30 inches.

## 4.2.5 Positive Present Net Value

Present Net Value (PNV) is the sum of the present value of the cash proceeds minus the sum of the present net value of the cash outlays required by an investment (Bierman and Schmitt, 1984). With zero taxes, the present net value of an investment may be described as the maximum amount a firm could pay for the opportunity of making an investment without being financially worse off. PNV is a valuable tool in the forest industry as well as in other industries. It provides a straightforward measure of the financial worthiness of undertaking a business venture, in the case of forestry, undertaking certain silvicultural prescriptions. PNV is especially useful when deciding between several prescriptions. As Davis and Johnson (1985) observe, the difficulty lies in doing it correctly. According to Davis and Johnson (1985) concerns center on the following:

1. "Have the future physical events of the project been adequately forecasted with respect to magnitude and timing?
2. Have the prices used to establish revenues and costs considered conditions of future markets, including possible changes in supply, demand, and inflation? Is the guiding rate adjusted correctly for inflation?
3. Does the guiding interest rate correctly reflect the owner's perception of the pure rate, inflation, and the riskiness of the project?
4. Has the problem situation been properly identified in the sense that the planning period is appropriate and the project chosen for the analysis is, in fact, feasible?
5. Does the schedule of events, costs, and revenues associated with the project include all those relevant to the decision maker, and in amounts actually affected by the project in question?"

In forestry, this analysis is particularly tricky because of the long time periods involved. Decisions may be very sensitive to the choice of the guiding interest rate and the handling of risk. Although interest rates are often adjusted for risk, Foster (1979) has effectively argued that if risk is best perceived as an accumulated "lump" at some future time, then adjusting the expected value of the return is preferable to including a premium in the interest rate for the purposes of comparing investments of different lengths.

In this project, a Present Net Value was used as a measure of the financial worthiness of a silvicultural prescription. Harvest volumes, harvest revenues and costs are assumed constant over time and are assumed to be known with certainty. For the purposes of the analysis, revenues and costs were taken from Cousar (1996), who did a similar analysis on the Sierra Nevada forests. Log revenues are a function of tree species and diameter. Timber harvesting costs include logging costs, road costs, slash disposal, NEPA (environmental analysis) costs, sale preparation and administrative costs. Logging costs are a function of MCF removed from the stand, average diameter of the harvest, and logging method. Equations for calculating tree cubic feet volumes were taken from Walters et al. (1985).

The guiding interest rate for this project was set at 6%. It is assumed to be net of inflation and indicative of the long term "before tax" interest rate used in private forest investments. Revenues and costs are also "before tax." These assumptions can be compared to the 4% discount rate used in evaluating long term forest investments by the USDA Forest Service.

## 4.3 Other Potential Goals: Alter Flow and Timing of Water

There are three direct physical hydrological effects that are affected by tree removal from the stand: changes in low flows, peak flows, and water yield.

- The difference in low flows between partial cuts and control is not very distinguishable (Reiter and Beschta, 1995).
- Peak flows depend on site-specific effects, elevation (water vs. snow), and proportion harvested. Harvest generally increases peak flow (Reiter and Beschta, 1995).
- In the South Umpqua National Forest changes in streamflow were investigated following shelterwood, patch cut and clearcut harvests. The authors found that the increases in the size of peak flows appeared to be related to the proportion of the watershed in which the soil was severely disturbed (Harr et al., 1979).

They also found that the largest absolute increases in water yield were in the winter, while the largest relative increases occurred in the fall and in the summer. However, their conclusion was that sustained yield forest management in southwestern Oregon does not have a sizable impact on increasing water yield. They did project that a large parent watershed would probably have yield increases of 4-6%. They stipulated that it appears that large yield increases due to harvesting are overshadowed by variability in normal flow from uncut and reforested stands.

These relationships suggest influences of harvesting on low flows, peak flows, and water yield. However, due to the lack of information on significant changes and clear relationships between thinnings and changes in waterflow, the alteration in flow and timing of water as a result of activities was not modeled.

# 5 THE PRESCRIPTION GENERATOR: PREMO

## 5.1 RLS-PATH and Look-Ahead Components

Several key elements of the RLS-PATH look-ahead algorithm are the length of the look-ahead period, the number of species and diameter classes, and the node interval. In order to use PREMO, the modeler must choose values for these variables. For the analysis that follows, a look-ahead period of 50 years was chosen. Since this is a young stand (see 3.1), a look-ahead of 50 years is important for realizing diameter growth. Four tree species are in the analysis: black oak (Quercus kelloggii), Douglas-fir (Pseudotsuga menzisii), Pacific madrone (Arbutus menzisii), and ponderosa pine (Pinus ponderosae). A two inch diameter class was used; a larger diameter interval would increase the averaging of diameters. A node interval of 50 TPA was chosen. Most classes have fewer than ten TPA, those with many often over 200. Also, using a 50 TPA node interval produces quicker solutions than a 1 TPA node interval.

## 5.2 Structure of the Prescription Generator: PREMO

Prescriptions for stand management are generated using the RLS-PATH algorithm (Yoshimoto et al., 1990) with a multi-stage look-ahead (Yoshimoto et al., 1988) of 10 periods. The goal programming objective function (Equation 5-1 displays the objective function for a given period) is to maximize the PNV minus the squared sum of the deviations from goal measures j=1,...,6. A scalar multiplier is applied to the PNV to regulate its influence on decisions. Goal measures are part of the objective function expressed as individual deviations from targets of measures 1 through 6. A goal emphasis defines the weights, targets, and which measures are used. A deviation is calculated only if the target level for the goal measure is not met.

$$\text{ObjectiveFunction}_{it} = MAX\left[p * \text{PNV}_{it} - \sum_{j=1}^{7} k_{ij} * \left(\text{Target}_{ij} - \text{Achievement}_{ijt}\right)^2\right]$$

$\text{ObjectiveFunction}_{it}$ = Decision Criterion for Goal $\text{Emphasis}_i$ at $\text{Period}_t$

$\text{PNV}_{it}$ = Present net value of activities at $\text{period}_t$ including value of the inventory at the end of the look ahead period

$\text{Target}_{ij}$ = Target of goal $\text{measure}_j$ for $\text{emphasis}_i$

$\text{Achievement}_{ijt}$ = Achievement of goal $\text{measure}_j$ for $\text{emphasis}_i$ at $\text{period}_t$

i = Goal emphasis

j = 1; Big trees for fish habitat (trees per acre)

j = 2; Vertical complexity

j = 3; Snags (trees per acre)

j = 4; Down woody debris (linear feet per acre)

j = 5; Fire behavior index

j = 6; Fire effects index

j = 7; Basal area threshold (square feet of basal area per acre)

$k_{ij}$ = Scalar multiplier for individual goal $\text{measures}_j$ for $\text{emphasis}_i$

p = Scalar multiplier for PNV

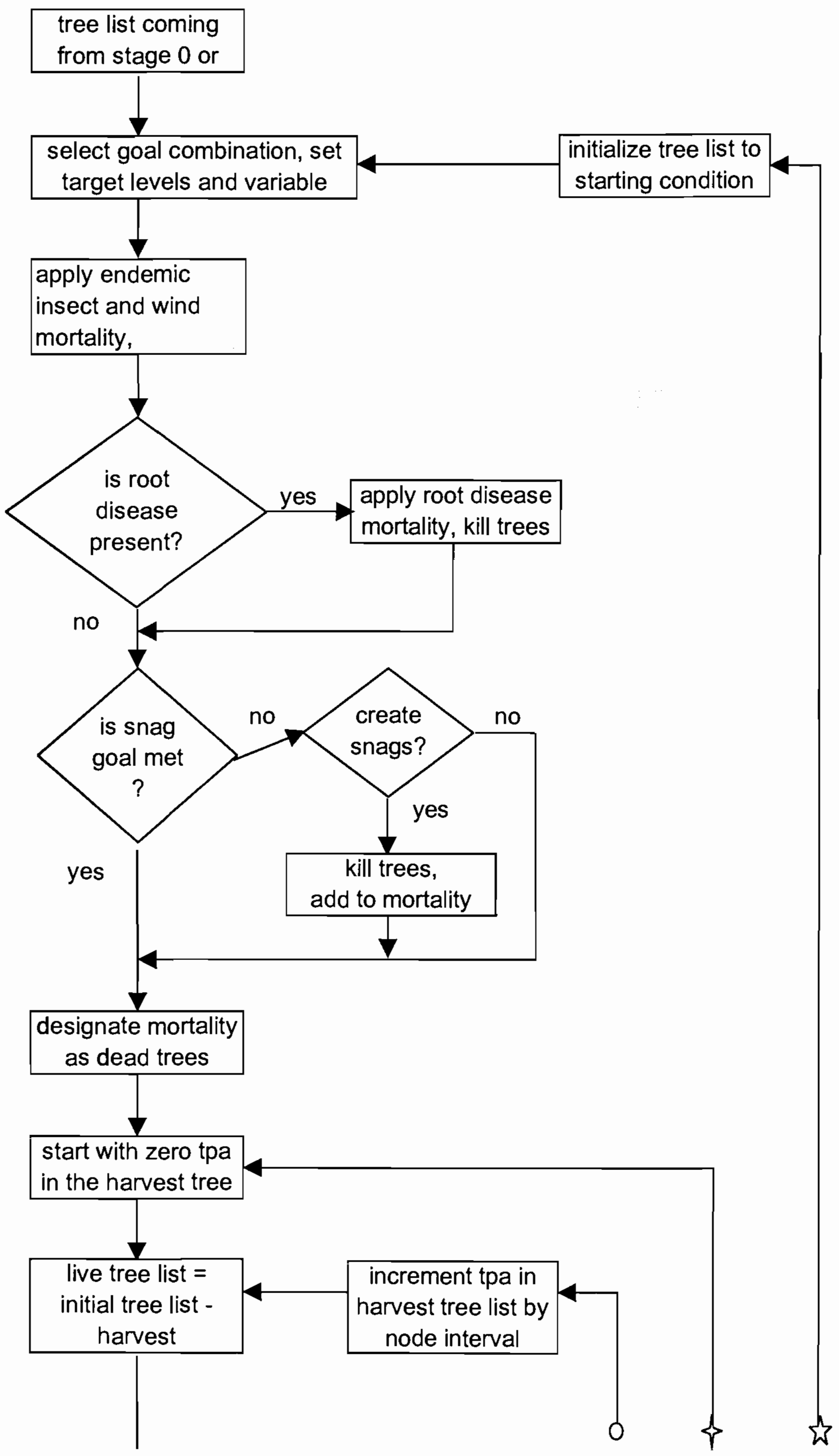
t = Time period

Equation 5-1. Objective Function

The path used in the optimization procedure is described in Figure 5-1. The prescription for each period defines which trees to remove and which to make into snags, in order to achieve the highest goal attainment at the current period and at the end of the look-ahead period. At the beginning of the flowchart, a starting tree list is recorded and the set of target levels and scalar multipliers that correspond with the first simulation (goal emphasis) are selected. Then the mortality from insects and from wind and root disease, if applicable, is applied. If this simulation includes the allowance to make snags, snags are created from trees with a DBH ≥ 15 inches, to meet the goal. Then the RLS-PATH algorithm is used to find the best prescription for this goal combination.

The cycle begins with zero trees in the harvest tree list. The trees in the harvest level are increased by the node interval until all of the trees in that diameter and species class are harvested. This process is continued for each diameter and species class. In this fashion, the cycle continues until adding one more diameter and species class does not increase the objective function. The harvest tree list is saved, and then the stand is grown for one period, mortality applied and the cycle begins again. This cycle occurs for the number of periods in the simulation. This whole process can be run for as many goal emphases as desired.

PRESCRIPTION GENERATION


tree list coming from stage 0 or
select goal combination, set target levels and variable
initialize tree list to starting condition
apply endemic insect and wind mortality,
is root disease present?
yes
apply root disease mortality, kill trees
no
is snag goal met ?
no
create snags?
no
yes
kill trees, add to mortality
yes
designate mortality as dead trees
start with zero tpa in the harvest tree
live tree list = initial tree list - harvest
increment tpa in harvest tree list by node interval

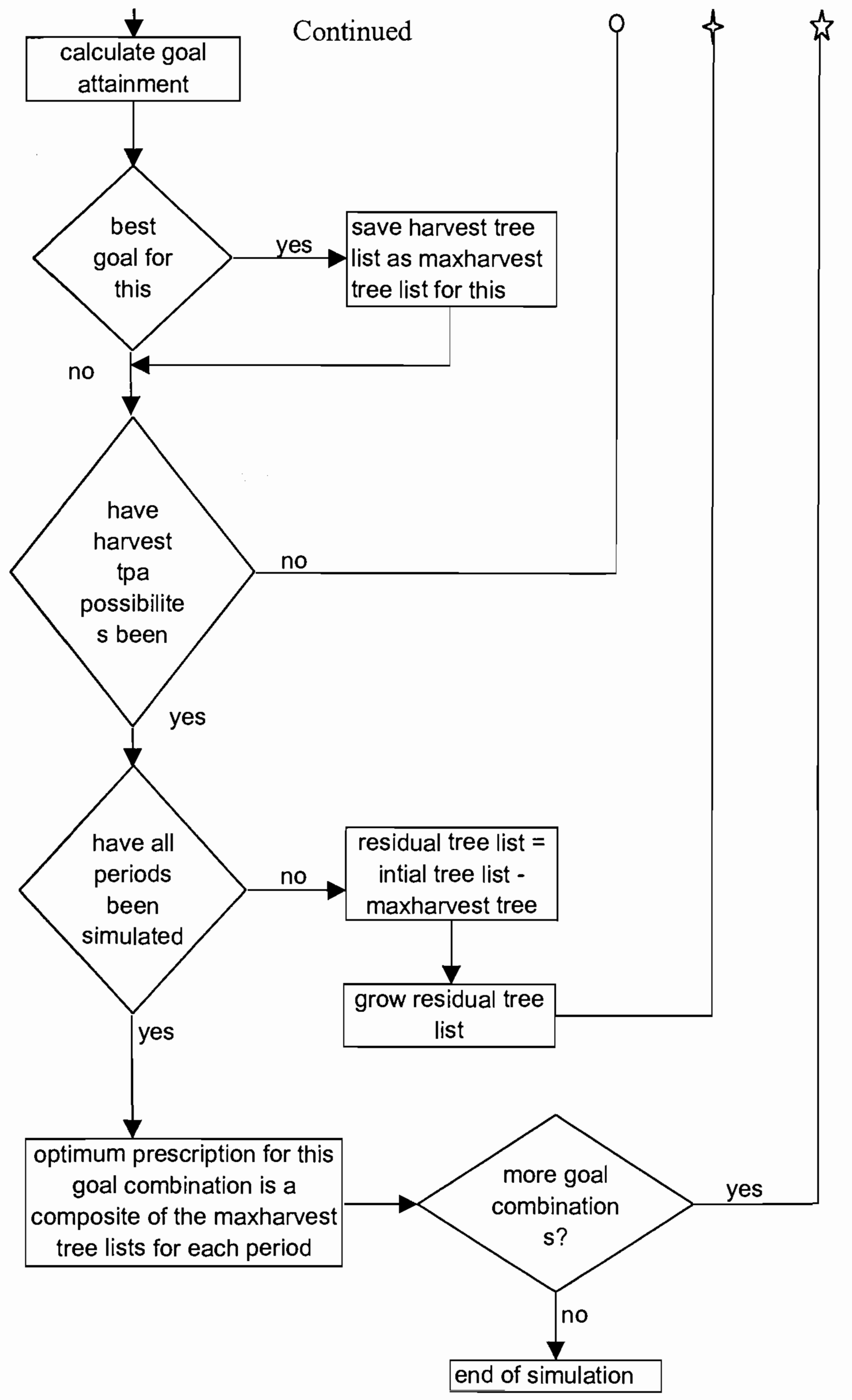

calculate goal attainment
Continued
best goal for this
yes
save harvest tree list as maxharvest tree list for this
no
have harvest tpa possibilite s been
no
yes
have all periods been simulated
no
residual tree list = intial tree list - maxharvest tree
grow residual tree list
yes
optimum prescription for this goal combination is a composite of the maxharvest tree lists for each period
more goal combination s?
yes
no
end of simulation

Figure 5-1. Flowchart of Prescription Generation

# 6 A SAMPLE ANALYSIS

In the analysis that follows, we utilized nine goal emphases (simulations) to create different prescriptions. The objective function varies depending on which goal measures are included. These goal emphases produce prescriptions that create different kinds of stand structure over time. PREMO is used to find efficient solutions for the combination of goal measures.

The stand to be modeled is a pine stand, without root disease, located in the pine/oak PAG, with a QMD between 9 and 15 inches and canopy closure above 60%. In the examples, PREMO finds the "best" prescription to achieve each of the nine goal emphases. A requirement was set on the harvest level, such that any harvest must remove at least 0.5 thousand cubic feet (MCF) in order for it to occur. The cost values will not be accurate for any harvest with a lower volume removal than this requirement. In PREMO, the harvest regime for a given period is determined. If this harvest does not exceed 0.5 MCF, then no harvest takes place in that period. The costs and revenues do not currently reflect values in the Applegate River Watershed and will need to be adjusted.

## 6.1 Goal Emphases

This thesis presents nine goal emphases that create at the most nine prescriptions, i.e., it is possible for two or more goal emphases to produce the same prescription. The first 6 goal emphases explore the stand potential for achieving target levels of goal measures that correspond to only one goal. For example, in the first goal emphasis, reduce fire risk, the only goal measures included in the objective function are the FBI and the FEI. The last three emphases begin to explore the achievement of several goals at the same time. Goal emphasis seven focuses on reducing risk from natural disturbances in the stand, while goal emphasis eight concentrates on habitat structure. In goal emphases six through nine a minimum harvest requirement of 0.5 MCF was set. Therefore, in a given period, if PREMO determines that a harvest is necessary in order to improve the achievement of goals, but the volume of that harvest does not exceed the MCF minimum, no harvest occurs in that period. The target levels for the goal measures in the emphases are listed in Table 6-1. The following list describes the focus of each goal emphasis:

1. Reduce fire risk
2. Reduce insect risk
3. Enhance fish habitat
4. Enhance wildlife habitat - complex structure
5. Enhance wildlife habitat - simple structure
6. Maximize PNV
7. Reduce fire and insect risk, maximize PNV
8. Enhance fish and wildlife – complex structure- habitat, maximize PNV
9. All goals, maximize PNV

| Goal Measure | Goal Emphasis | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | Fire | Insect | Fish | Wild life - comp lex | Wild life - simple | PNV | Fire, Insect, PNV | Fish, Wild life, PNV | All, PNV |
| Big trees for fish | X | X | ≥5 | X | X | X | X | ≥5 | ≥5 |
| Vertical Structure | X | X | X | ≤50 | ≥100 | X | X | ≤50 | ≤50 |
| Snags | X | X | X | ≥5 | ≥5 | X | X | ≥5 | ≥5 |
| Linear Feet DWD | X | X | X | ≥120 | ≥120 | X | X | ≥120 | ≥120 |
| Fire Behavior Index | ≤31 | X | X | X | X | X | ≤30 | X | ≤30 |
| Fire Effects Index | ≤30 | X | X | X | X | X | ≤30 | X | ≤30 |
| Basal area thresholds | X | ≤80 | X | X | X | X | ≤80 | X | ≤80 |

Table 6-1. Goal Target Levels.
"X" means that the goal was not used.

## 6.2 Goal Achievement

In the graphs that display the values for the goal measures by period, the values are displayed as would be measured in the stand just before harvest at the end of the 5-year growth period. Another possibility is to display the values after harvest. This thesis displays the values at the end of the growth period, but before harvest, because this usually shows the largest deviation from the goal over time that will occur. If the values were displayed after harvest, the deviations from the goal would usually be smaller since the action had just been taken to reduce the deviations caused by growth in the previous growth period. This will be true in the case of goal emphases which have a single goal measure, and will usually be true for goal emphases that have multiple goal measures, but not always.

### 6.2.1 Fire Hazard

The minimization of fire hazard is a goal in emphases 1, 7, and 9. Therefore, the two indices, FBI and FEI are included in these emphases as goal measures. The target levels remained the same in each goal emphasis. The target level is to have an FBI and FEI less than 30. An FBI $\leq 30$ permits a lower flame length and higher crown bulk density, or a higher flame length and lower crown bulk density, but no torching. Any stand where torching would occur will have an FBI > 30. The FEI is set at 30 because among some species, even large trees will have 20% basal area mortality in a fire, however small trees have close to a 90% basal area mortality. An FEI=0 or near zero would force the stand to have almost no trees at all.

### 6.2.2 Goal Emphasis (#1): Fire Hazard Reduction

The FBI before harvest by period is displayed in Figure 6-1. In periods 1, 2, 10, 12, 14, 18, and 20 the FBI is above the target level. In each of these instances torching in the stand is possible. In the all of these instances, this exceeding of the goal is due to regeneration from the previous harvest. Figure 6-2 shows when harvest occurs. Regeneration is not put into the model until the optimum harvest level has been chosen. As it is not a part of the look-ahead, the effect of regeneration on the ability to meet goals is not considered. In all other periods the prescription produced an FBI value below the target level. Future versions of this model should perhaps reevaluate when the regeneration is considered.

For this goal emphasis only, the FBI after harvest is displayed for comparison in Figure 6-1. After harvest it is possible to reduce the FBI below the target level for all but two periods. The two times where the FBI remains above the target are due to the RLS-PATH process. Harvesting from one diameter class alone does not improve the objective function enough to continue the process. In the "grow only" scenario (not displayed), the FBI reaches 46.9 by the 5th period and increases to 85 by the 18th period, where it remains.

As can be seen in Figure 6-1, it is possible for the FBI to increase after harvest. This is due to an increase in crown bulk density. Crown bulk density is assigned based on vegetation class, therefore, if the harvest shifts the stand into a vegetation class with a higher crown bulk density, the FBI can increase. This change in crown bulk density is an artifact of our data set and could be changed in the future.

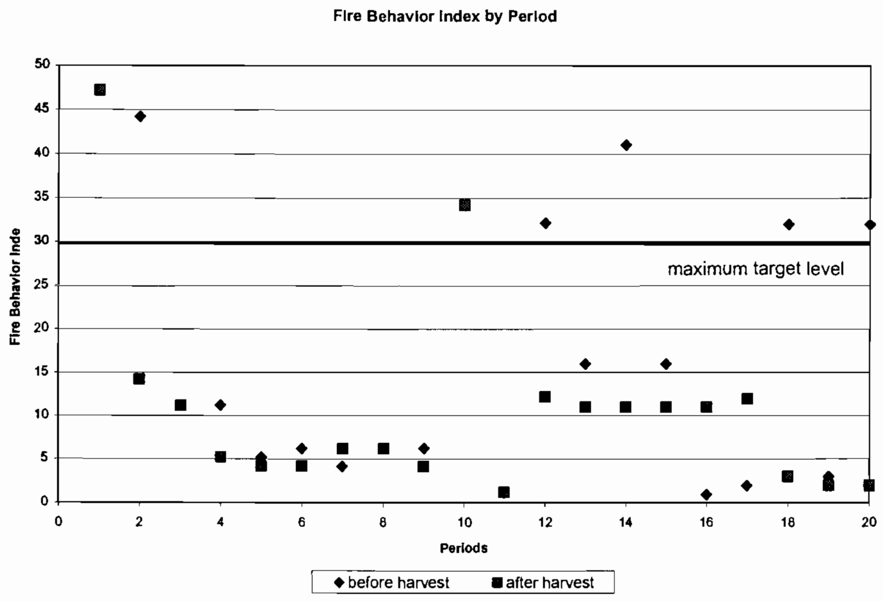


Figure 6-1. FBI by Period before Harvest for Goal Emphasis #1.

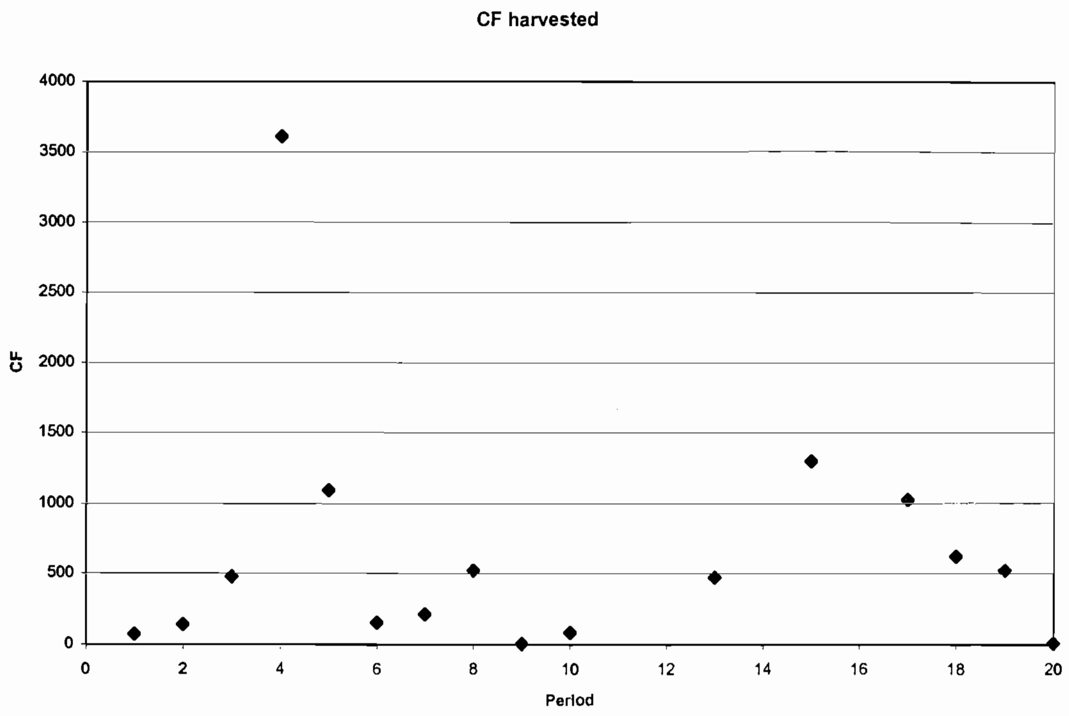


Figure 6-2. Cubic Feet Per Acre Harvested per Period for Goal Emphasis #1.

The periods in which the FEI is above the target level (10 and 13) also correspond to periods of regeneration (Figure 6-3). The basal area that would die in a fire is higher for little trees. However, the FEI is not far above the target level. In comparison, when no trees are harvested, the FEI reaches a value of 92.16 by the 10th period and remains high.

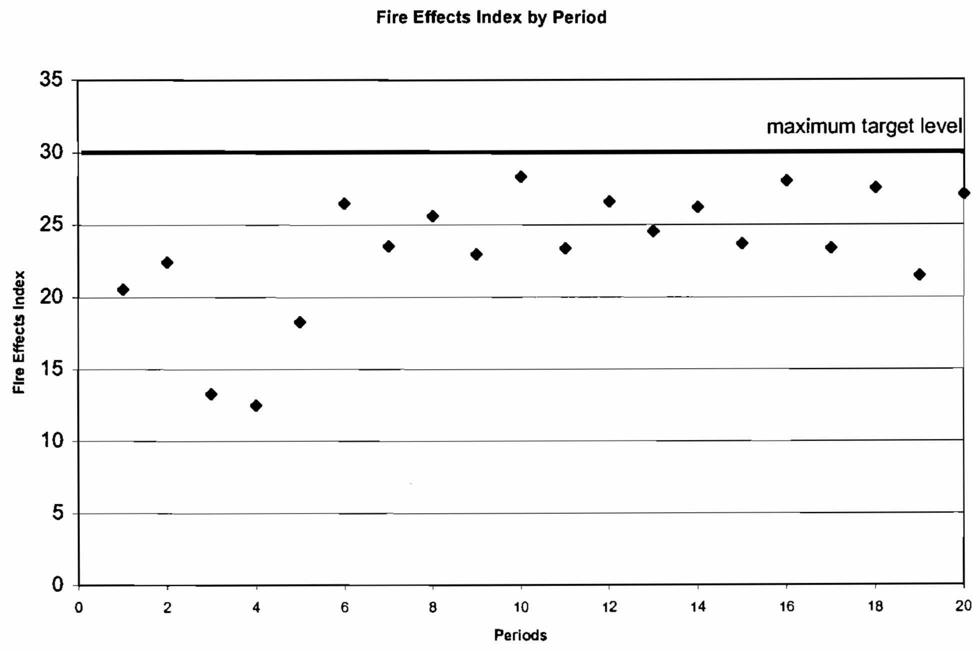


Figure 6-3. Fire Effects Index by Period before Harvest for Goal Emphasis #1.

## 6.2.3 Goal Emphasis (#7): Stand Hazard Reduction

The FBI is greater than the target level for 7 of the 20 periods. In each of these periods torching is possible in the stand. This prescription only schedules five harvests over the planning horizon (Figure 6-5). The minimum harvest requirement of 0.5 MCF, associated with the PNV goal, prevents the model from removing the smaller trees which are acting as fuel ladders, causing torching. The minimum harvest requirement prevents the removal of smaller trees, which have little to know volume, unless there is enough volume in larger trees scheduled for harvest to meet the requirement. The FBI per period for this goal emphasis is shown in Figure 6-5. This goal emphasis has a much lower basal area per acre over time than goal emphasis #1, due to the "limit insect hazard goal". As a result, there is more regeneration in this open stand than in goal emphasis #1 and torching can occur.

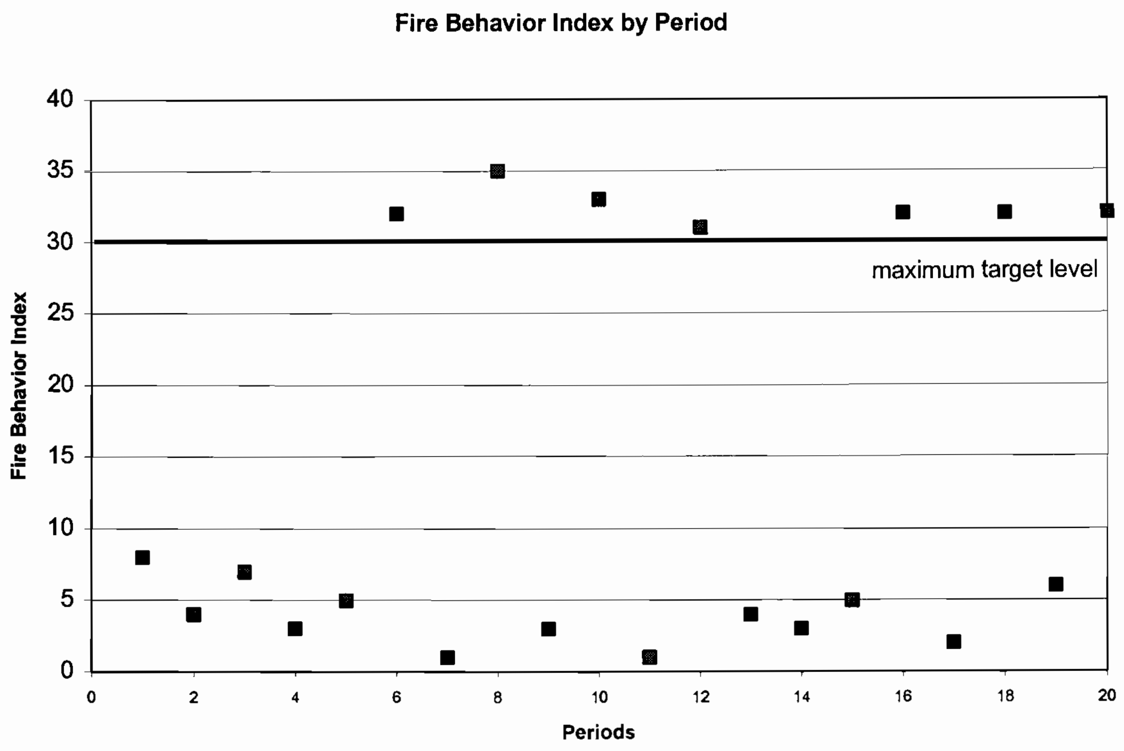


Figure 6-4. FBI by Period before Harvest for Goal Emphasis #6.

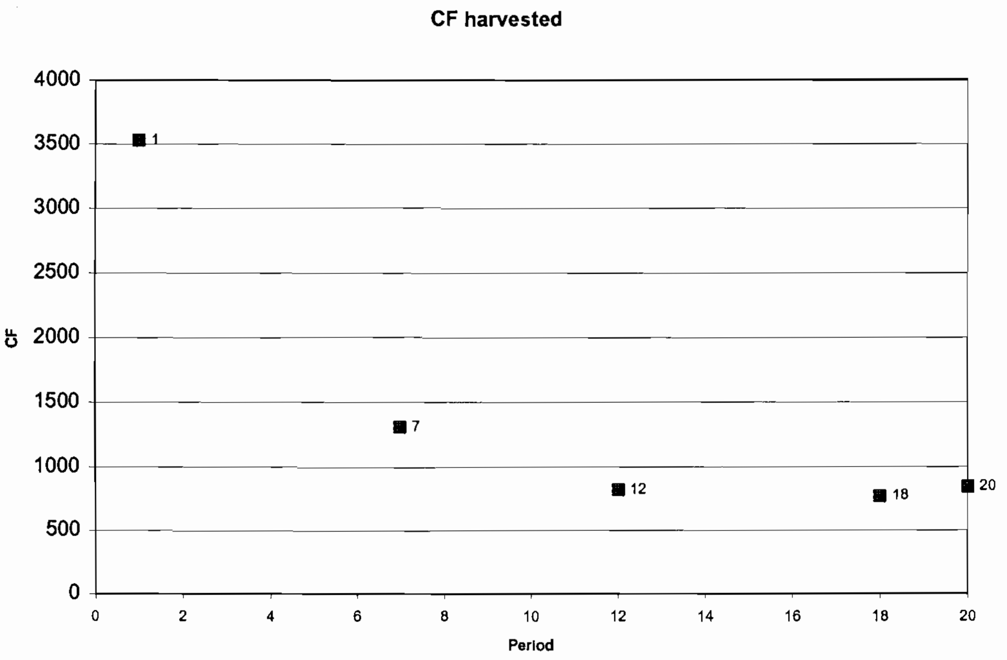


Figure 6-5. Cubic Feet harvested by Period for Goal Emphasis #7.

The FEI is above the target level in 3 periods (Figure 6-6). None of these periods were preceded by a harvest, which could have removed the smaller trees and thus reduce the basal area mortality of the stand associated with fire.

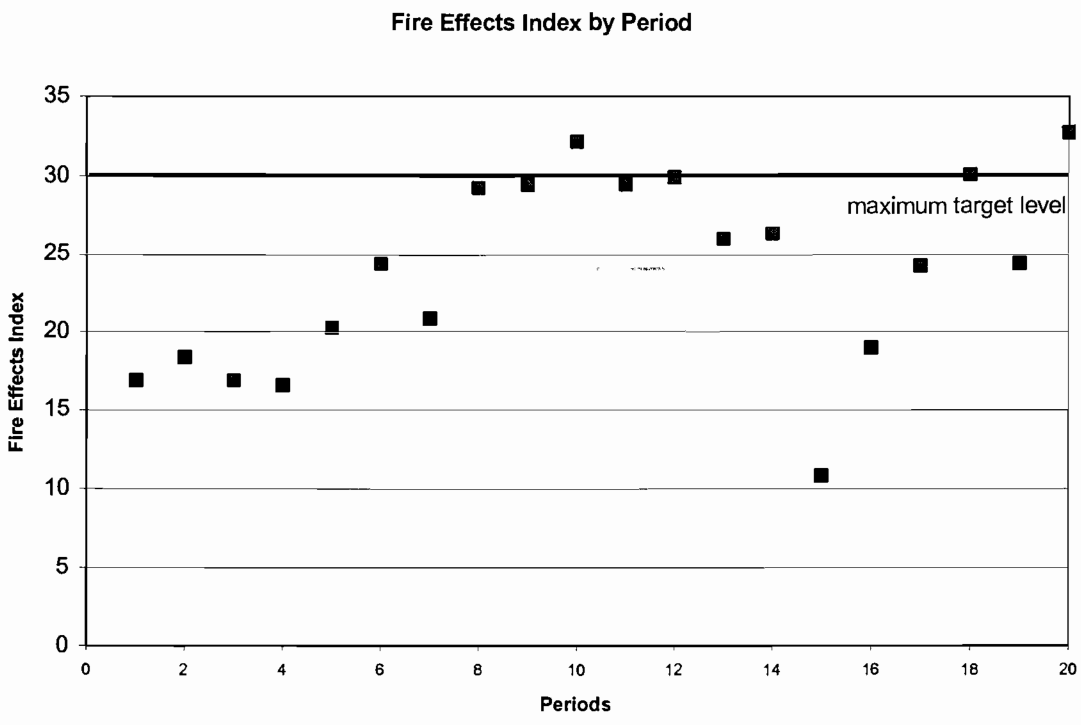


Figure 6-6. FEI by Period before Harvest for Goal Emphasis #7.

## 6.2.4 Goal Emphasis (#9): All Goal Measures

The FBI is above the target level for periods 6, 10, 12, 16, and 18 (Figure 6-7). In each of these periods torching occurs which is responsible for 30 points of the FBI. Each of these periods would have benefited from a removal of the trees causing the torching in the period when the goal was not met, however, due to the minimum harvest volume requirement this only happened in periods 12 and 18.

Torching results from the regeneration that is added from harvests in periods 1, 7, 9, 12, and 18 (Figure 6-5). It is possible to have no torch potential in the stand, even when regeneration occurs, as long as the crowns of the overstory are much higher than the regeneration so that the regeneration does not act as a fuel ladder. This situation occurs after the harvest in period 1, resulting in a low FBI for a few periods thereafter.

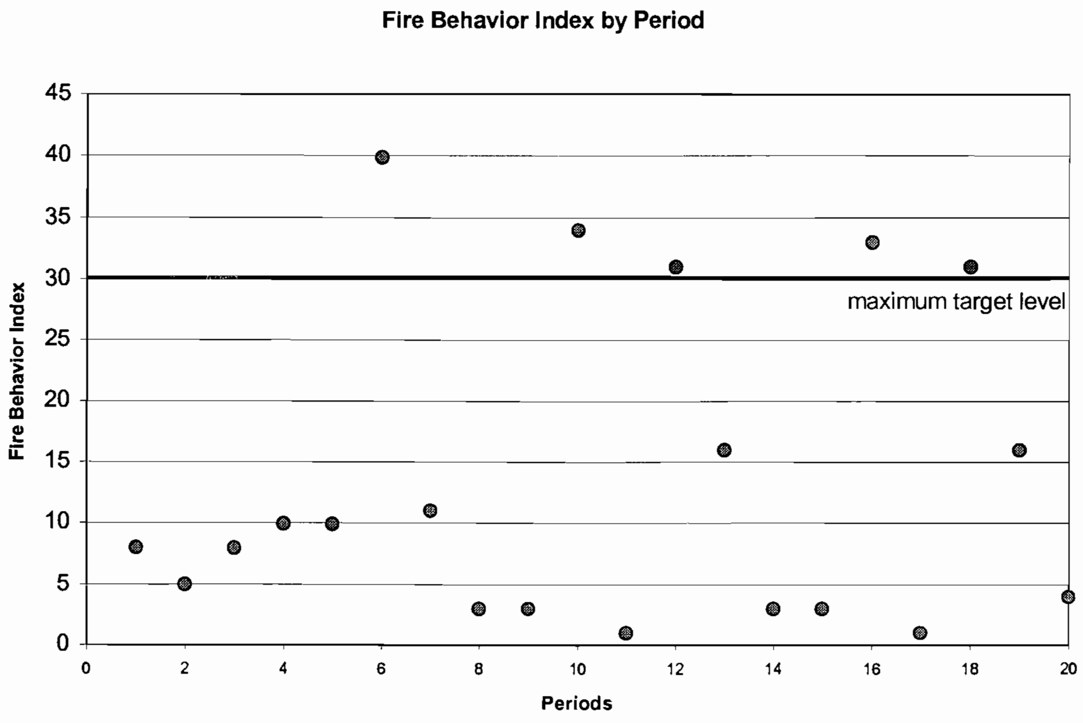


Figure 6-7. FBI by Period before Harvest for Goal Emphasis #9.

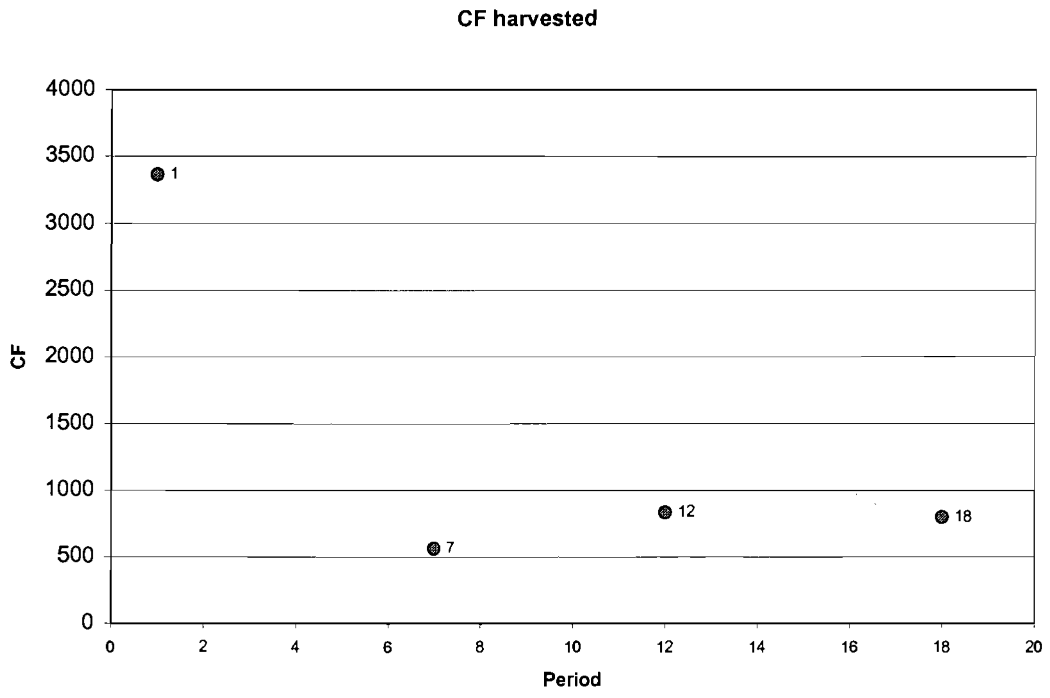


Figure 6-8. Cubic Feet Harvest per Period for Goal Emphasis #9.

The FEI is above the target level in only four periods (Figure 6-9). The first two occurrences above the target level are due to regeneration from the first harvest that is not removed until period 7. The last two occurrences above the target level are a result of regeneration and low basal area, upon which smaller trees have greater effect. The trend that can be seen in the FEI differs from that of goal emphasis #7. After the harvest in period 7, goal emphasis #9 has far more large trees and fewer smaller trees, than does goal emphasis #7. As a result, the FEI is higher for goal emphasis #7, as the smaller trees have a larger effect on the FEI.

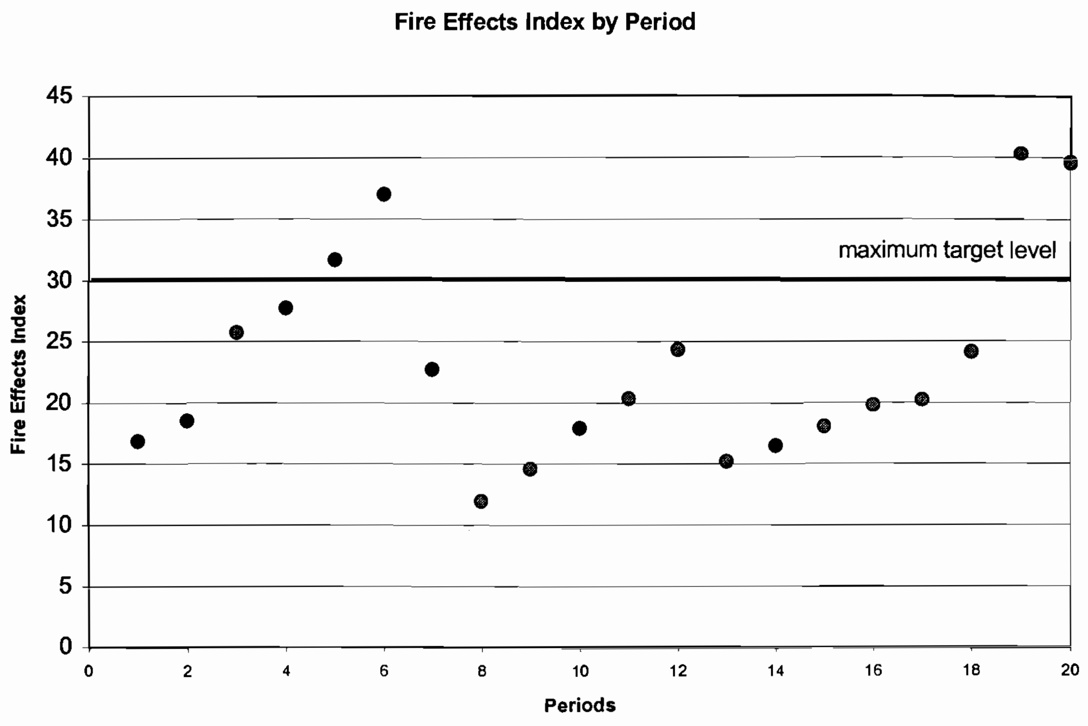


Figure 6-9. FEI by Period before Harvest for Goal Emphasis #9.

## 6.2.5 Wildlife Habitat

There are three goal measures for wildlife habitat: vertical structure, snags, and DWD. These goal measures were used in emphases 4, 5, 8, and 9. Goal Emphasis #5 is the only one focusing on creating simple stand structure. All others have a goal of complex stand structure.

There are two target levels for vertical structure, one corresponding to a complex stand, and the other to a simple stand. Each of these targets for vertical structure is a difficult one to achieve because the stand is not static and trees may be simultaneously removed for snags. In Goal Emphasis #4, where the goal is complex structure, the vertical structure in periods 8 of the 20 periods is not complex, however, it is very close to the target level. In the even-numbered periods, the complexity is disrupted by regeneration that comes in at the beginning of the period following harvest. In Goal Emphases #8 the stand becomes more simple in periods 9-13, 15-17, 19 and 20. The goal of creating snags also makes it difficult to achieve this goal. This goal measure does not work very well when there are height classes that have no trees at all, since the goal measure tries to keep the same number of crowns in each height class. Because of the large trees for the fish habitat goal, there are some larger diameter trees, however, there are several diameter classes in the mid-diameter range, which causes a two cohort strata which has simple stand structure. Because the large trees that were to enhance the structure for fish habitat were being turned into snags, an upper limit of 30" DBH was set for snag creation. This caused the two-cohort stand, because snags came from the mid-diameter range of the stand. Finally, in goal emphasis #9 the stand structure is complex in 12 of the 20 periods and the stand structure is simple in 8 of the 20 periods where harvest and snag creation has created a two-cohort stand.

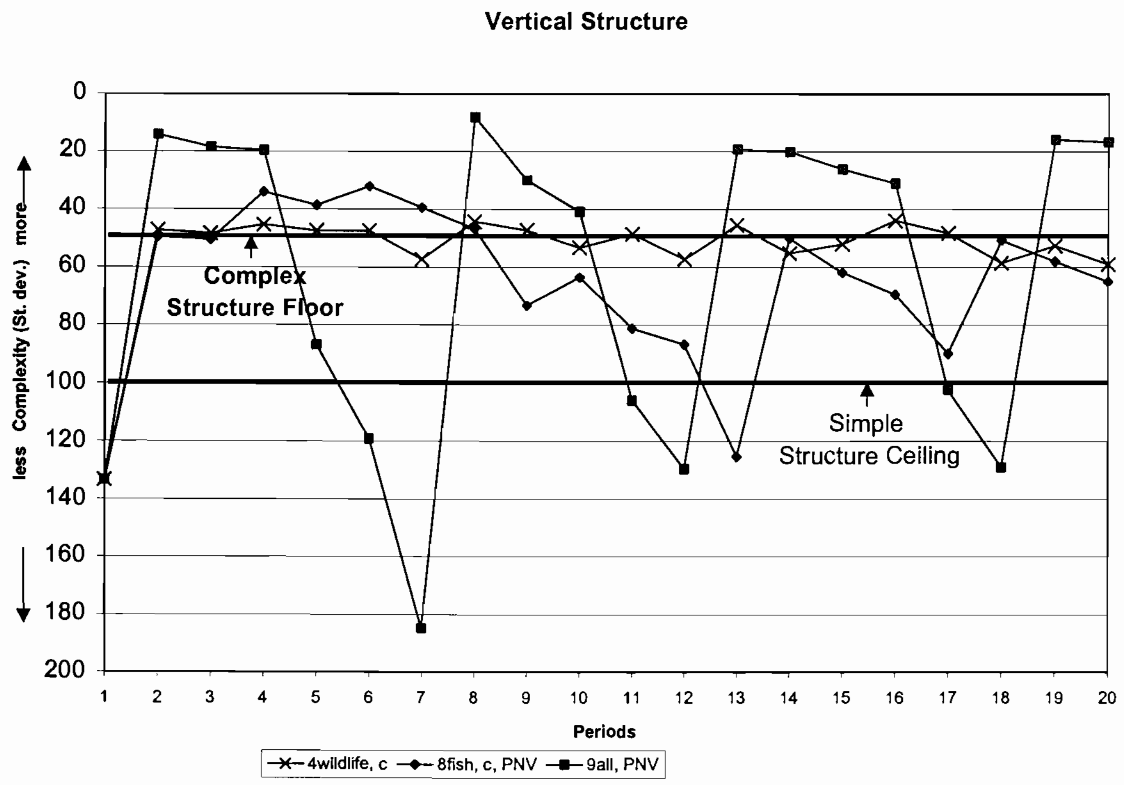


Figure 6-10. Vertical Structure before Harvest for Goal Emphases #4, #8, and #9.

The goal for Goal Emphasis #5 is to produce a stand that has simple structure. The stand structure by period can be seen in Figure 6-11. The target level was satisfied for almost all periods.

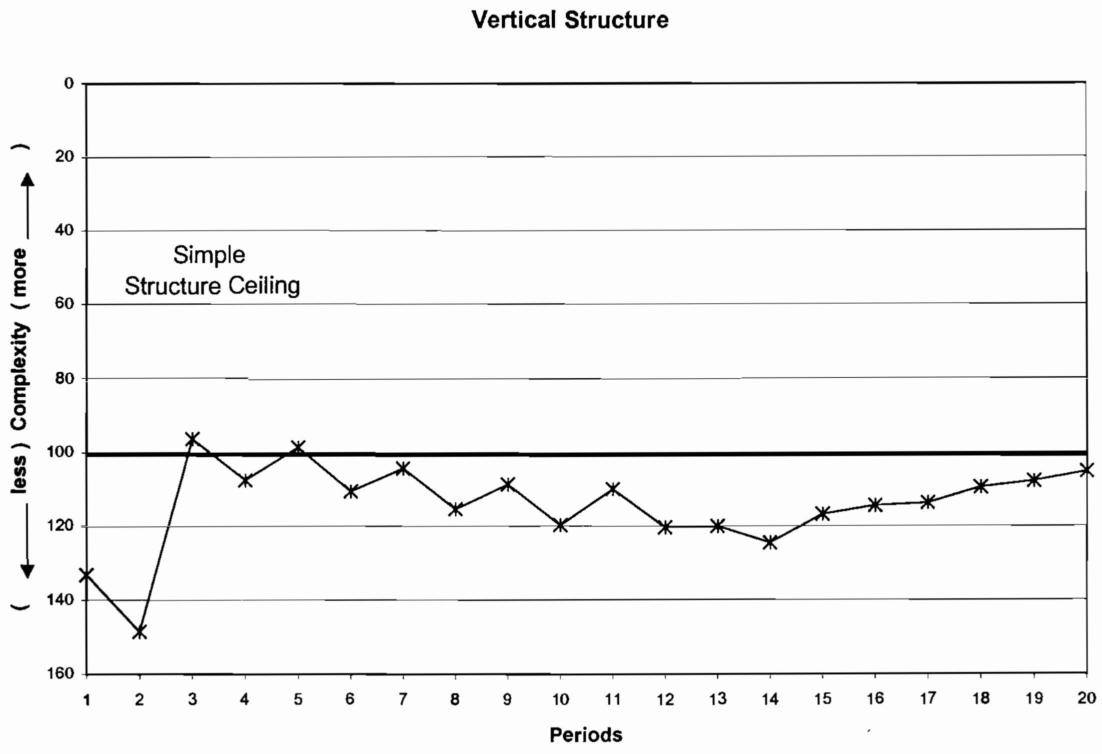


Figure 6-11. Vertical Structure before Harvest for Goal Emphasis #5.

The number of snags per period is displayed in Figure 6-12. Snags are created in the first period in order to reach the snag goal per acre. The graph is pre-harvest, so it displays the number of snags in a period before any are created. Thus, when snags are created, they do not appear until the next period. Snags decay and fall over time. Snags which are created begin at Class I and move toward Class V. Once a snag moves beyond Class V it is considered to be disintegrated. Snags which reach Class V are not counted any more. If the snag goal is not met two periods in a row, the stand does not have any trees from which to make snags. It is possible to be below the snag goal for one period due to snags falling, but by the second period the program will have created the necessary snags to achieve the snag goal IF there were available trees from which to create snags. In goal emphases #5 and #8 more snags are created as trees grow to be large enough from which to make snags. For instance, in the prescription for goal emphasis #8, a group of trees reaches a DBH greater than 15 inches and more snags are created, however in the prescription for goal emphasis #9, there are no more large trees in the rest of the planning horizon.

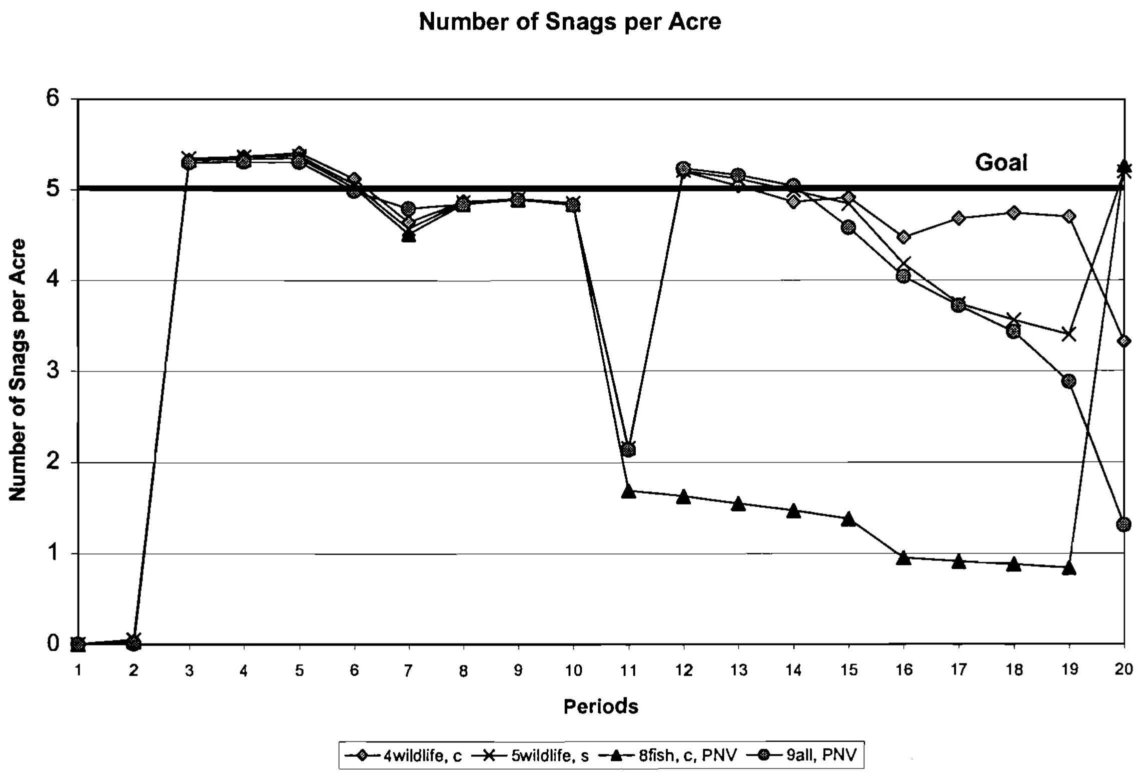


Figure 6-12. Number of Snags before Harvest for Goal Emphases #4, #5, #8, and #9.

The goal for 120 linear feet of DWD per acre is never reached. In PREMO there is no mechanism available that allows the creation of DWD if it is short of the target. However, this could be added in a way similar to the creation of snags. A discrepancy arises because snags are accepted with a DBH of 15 inches, however, DWD must have a large diameter of at least 16 inches. In this tree list, one group of snags that is created in the first period has a DBH less than 16 inches, and as a result any DWD coming from this tree record does not count as DWD. However, there are larger snags created by period 11 and an increased presence of DWD can be seen in some of the goal emphases after this period as a result of some of these snags falling. In the snag model there is a decay lag and this plot for the sample analysis did not have any DWD in the original plot data so the result is no DWD for the first five periods.

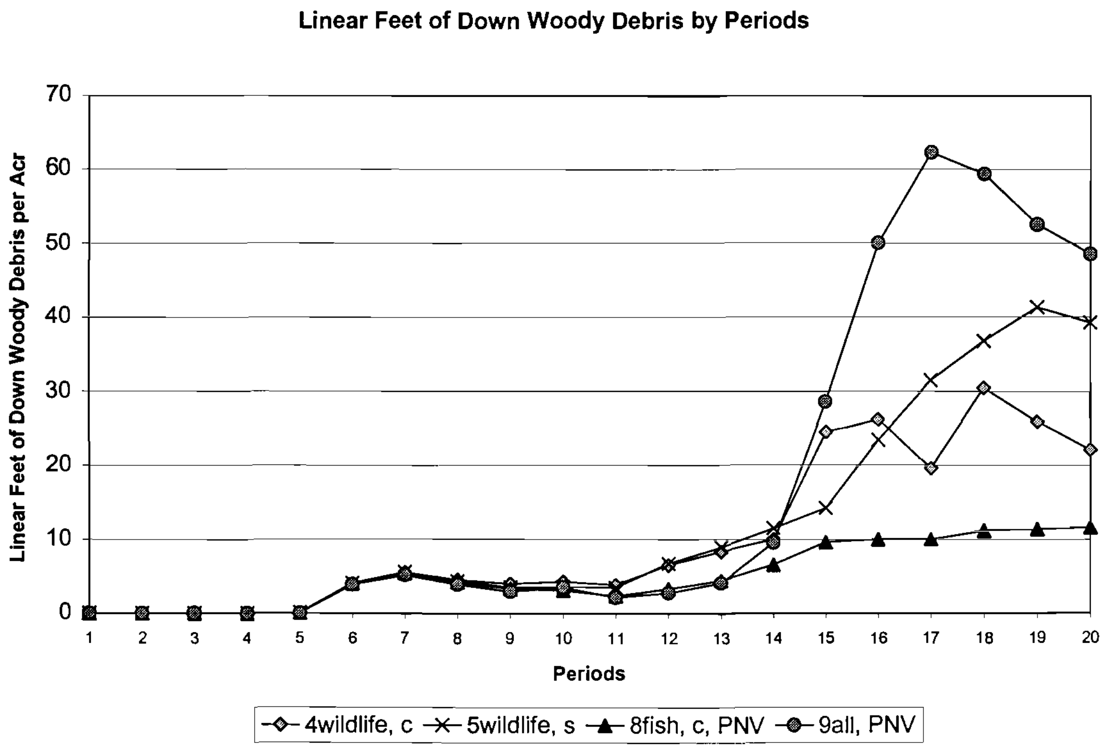


Figure 6-13. Linear Feet of DWD by Periods for Goal Emphases #4, #5, #8, and #9.

## 6.2.6 Insect Hazard

For a stand with no white fir component in the pine/oak PAG, the basal area threshold is 80 sqft/acre. In the landscape simulator, stands above this threshold will experience Douglas-fir and pine mortality in drought years. Insect hazard was a goal measure in emphases #2, #7, and #9. In Goal Emphasis #2, the basal area of the stand was kept below the threshold for all periods (Figure 6-15). The basal area is kept well below the target level so that even at the end of the look-ahead period the basal area without harvest is below the target level.

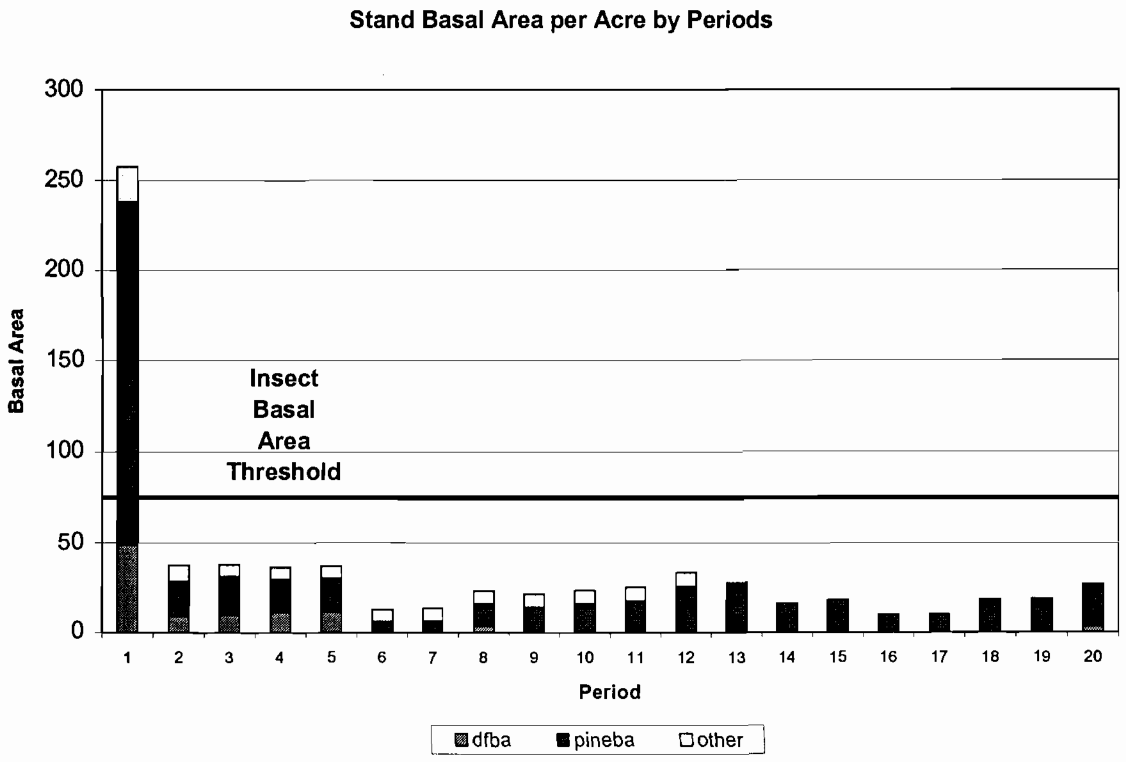


Figure 6-14. Stand Basal Area before Harvest per Acre for Goal Emphasis #2.

The basal area in goal emphasis #7 only exceeds the basal area threshold in 4 periods after the initial harvest (Figure 6-15) because of a minimum harvest volume requirement. Up to the 12th period goal emphasis #9 maintains a similar stand basal area per acre as emphasis #8, however, later it maintains greater basal area due to the wildlife and fish habitat goals and exceeds the threshold because of the minimum harvest volume requirement.

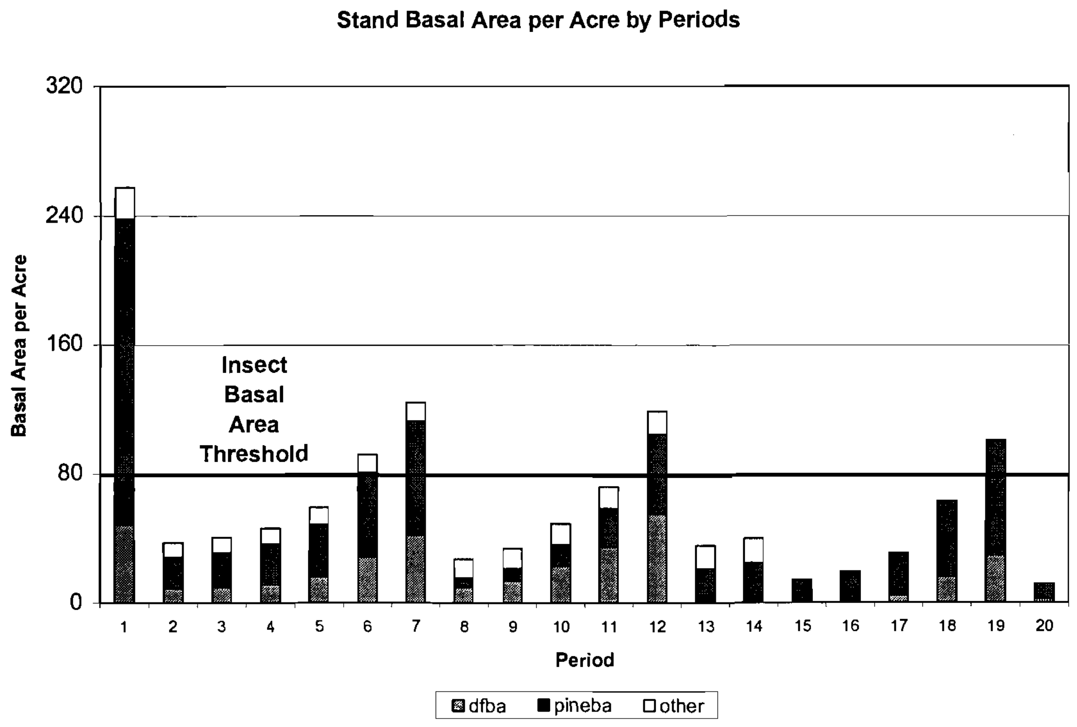


Figure 6-15. Stand Basal Area before Harvest per Acre for Goal Emphasis #7.

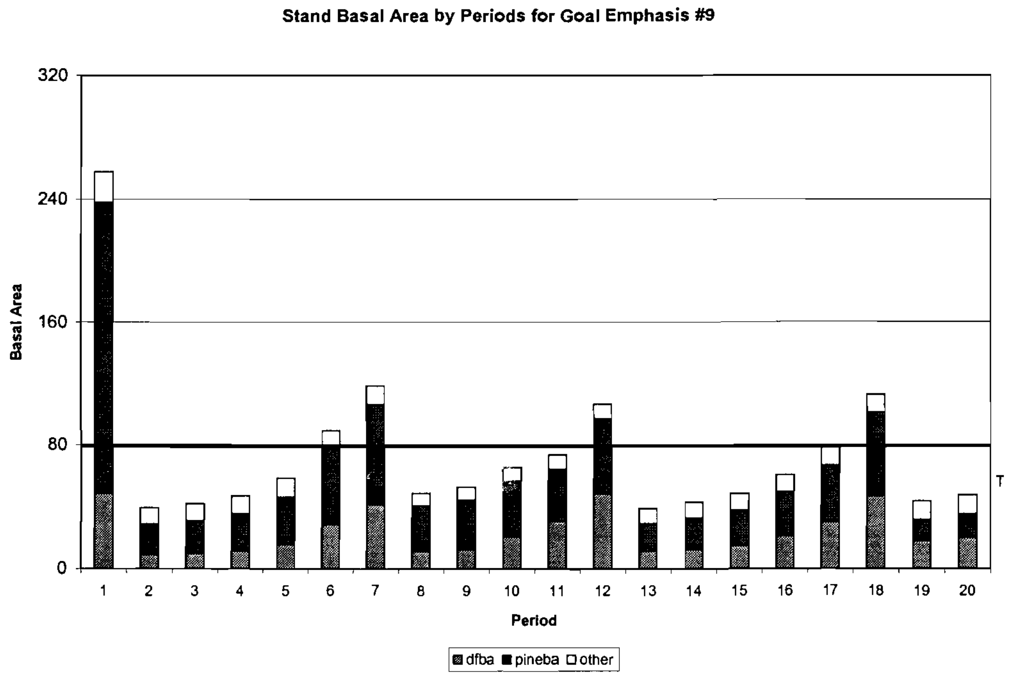


Figure 6-16. Stand Basal Area before Harvest per Acre for Goal Emphasis #9.

## 6.2.7 Fish Habitat

As the stand chosen for simulation is not in, but near, a riparian area, the only goal measure used for fish habitat is number of large trees per acre. If the stand had been in a riparian area, there would be an additional goal measure of canopy closure. The goal measure of large trees per acre was used in emphases #3, #8, and #9. In the first few periods it is not possible to meet the goal, since the analysis stand had a low original stand QMD and there are no trees with a DBH greater than 30 inches. In goal emphasis #3, as the trees grow, the goal is met. However, in the other two goal emphases, trees that have the potential of being recruited for the large tree component are removed before they reach that size, due to other goals.

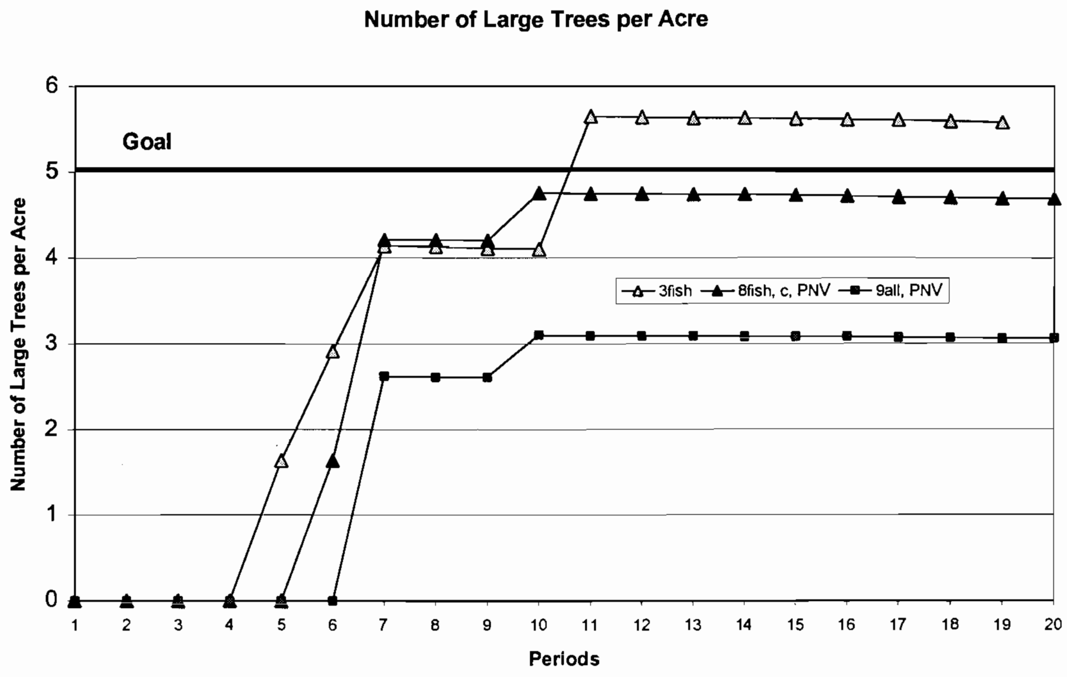


Figure 6-17. Number of Large TPA before Harvest for Goal Emphases #3, #8, and #9.

## 6.3 Summary

### 6.3.1 Present Net Value

The total value per goal emphasis is displayed in Figure 6-18. The largest value is produced in the goal emphasis focusing on only PNV. However, all other goal emphases also provide a positive PNV. The PNV is only shown for those four goal emphases that included PNV as a goal. These values will likely change after incorporation into the landscape model, as the PNV goal emphasis has higher fire and insect risk and the returns from activities might not be realized. In the landscape model, trees die because of periodic disturbance and occurrences thereof will be higher in the PNV goal emphasis in goal emphases 8 and 9, which are more resistant to fire and insect attacks, and thus the harvests in the PNV goal emphasis will be reduced, lowering the PNV.

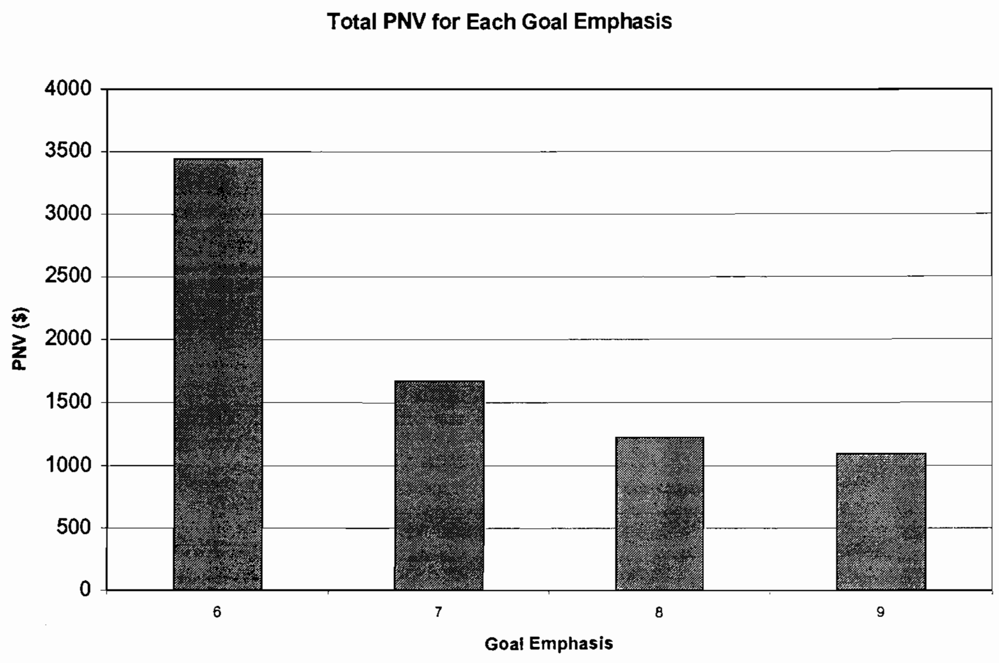


Figure 6-18. Total Value for Each Goal Combination.

## 6.3.2 Activities and Vegetation Class

The cubic feet harvested by period can be seen in Appendix C. The trees per acre that were made into snags are displayed in Appendix D.

The different goal emphases influence the stand structure and the species composition over time. As the stand grows and activities are carried out, the stand's species composition and structure changes; therefore, the classification of the stand changes as well. These stand dynamics can be seen in Table 6-2. The first number of the three numbers is the cover type, the second is the QMD, and the third is percent cover (0=<60% cover, 1=>60% cover.)

The cover type is determined according to the following rules, where hardwood, conifer, and pine represent the percent composition of the included species of the upper canopy:

1. Conifer/Hardwood: hardwood>30%, conifer>30%
2. Deciduous Hardwood: hardwood>30%, conifer<30%, deciduous hardwood>50%
3. Evergreen Hardwood: hardwood>30%, conifer<30%, deciduous hardwood<50%
5. Mixed Conifer: hardwood<30%, pine<50%
7. Pine: hardwood<30%, pine>50%

The QMD number is assigned according to the following distribution depending on into which category the adjusted QMD fits:

1. 5"-9"
2. 9"-15"
3. 15"-21"
4. 21"-25"
5. 25"-32"
6. 32"+

| Period | Goal Emphasis | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
| 1 | 721 | 130 | 721 | 731 | 721 | 721 | 130 | 530 | 13 |
| 2 | 730 | 130 | 721 | 731 | 721 | 721 | 130 | 730 | 74 |
| 3 | 721 | 140 | 721 | 730 | 721 | 520 | 130 | 730 | 74 |
| 4 | 720 | 140 | 731 | 730 | 721 | 520 | 720 | 730 | 12 |
| 5 | 120 | 230 | 731 | 730 | 721 | 521 | 720 | 530 | 12 |
| 6 | 520 | 120 | 731 | 730 | 720 | 711 | 520 | 520 | 51 |
| 7 | 520 | 120 | 530 | 730 | 521 | 721 | 120 | 720 | 75 |
| 8 | 120 | 120 | 521 | 530 | 521 | 721 | 120 | 520 | 13 |
| 9 | 120 | 230 | 521 | 530 | 521 | 711 | 120 | 120 | 13 |
| 10 | 510 | 230 | 521 | 730 | 521 | 511 | 510 | 120 | 13 |
| 11 | 520 | 230 | 530 | 530 | 520 | 521 | 520 | 120 | 52 |
| 12 | 521 | 720 | 230 | 530 | 521 | 521 | 120 | 521 | 56 |
| 13 | 520 | 710 | 150 | 130 | 520 | 120 | 120 | 130 | 56 |
| 14 | 521 | 720 | 330 | 130 | 521 | 120 | 120 | 120 | 12 |
| 15 | 520 | 700 | 330 | 130 | 521 | 120 | 120 | 120 | 12 |
| 16 | 520 | 710 | 330 | 130 | 521 | 121 | 120 | 120 | 52 |
| 17 | 510 | 710 | 330 | 121 | 521 | 120 | 520 | 330 | 52 |
| 18 | 720 | 710 | 330 | 131 | 521 | 120 | 130 | 340 | 56 |
| 19 | 710 | 710 | 340 | 131 | 521 | 120 | 130 | 340 | 56 |
| 20 | 510 | 720 | 340 | 131 | 531 | 121 | 130 | 121 | 12 |

Table 6-2. Vegetation Class by Period, for Each Goal Emphasis.

# 7 DISCUSSION

This thesis had two major objectives. The first was to develop a multi-goal prescription generator for the forests of the Applegate River Watershed. Then the ability of the model to meet goals was evaluated by a sample stand analysis.

## 7.1 Prescription Generation

A multi-goal prescription generator, PREMO, was developed to create prescriptions for the forests of the Applegate River Watershed, to be used in the ARWFSP landscape simulator. Following are the three subobjectives for the creation of this model, how they were accomplished, and improvements that might be made.

### 7.1.1 Goal Measures

A key part of this project was to develop goal measures from vague goal statements. This was an enjoyable and challenging part of the project in working with different project scientists to develop creative measures that would be able to describe the stand in terms that were important for each goal measure. Target levels were discovered in a similar manner, from scientists and their research and findings. This process resulted in a successful development of goal measures and targets.

A remaining issue lies in the measurement of the vertical structure for the wildlife habitat goal. The vertical structure goal might not adequately produce complex and simple stands. For instance, in stands that have an initial simple structure, the model creates a complex stand by harvesting trees from each height class until all height classes have the same number of crowns in them. The problem with this method is that the height class with the fewest crowns determines the stocking of the stand.

Two additional issues with the wildlife habitat goals stand out from the results. The creation of snags is not an actual goal measure within the RLS-PATH section. Currently, snags are either created, or not created. When they are created and if possible, PREMO seeks to meet the target level for snags in every period. There is no deviation from this goal if there are sufficient number of trees for snag recruitment. In the emphases with multiple goals this will cause conflict in limiting creation of large live trees.

Also, the most dynamic changes in the stand due to the prescriptions seem to be in the changes in cover type. Species composition of the stand appears to be more affected than structure, at least in this analysis, and perhaps needs to be added as a goal measure. Finally, the creation of DWD might also be considered as a goal.

### 7.1.2 PREMO Growth Component

Translating the FVS model and calibrating the growth component of PREMO took a significant part of the time spent building the prescription generator. The Klamath Mountains Variant has 312 subroutines, and the first challenge was to out which are used for growth and which to translate. Furthermore, code that somebody else has written is very difficult to decipher, due to unknown variables and processes. As mentioned earlier, the code for PREMO is in Appendix A: Growth Subroutines.

The assessment of the PREMO's ability to achieve results the same as those of FVS is described in 3.2 above. However, whether PREMO represents how the forests in the Applegate River Watershed grow is another question. The growth in PREMO, when not affected by harvest, may not be realistic. The absence of episodic mortality results in larger basal areas being carried in the stand in some goal emphases that seem higher than experienced. As the landscape simulation is performed, these basal areas will likely be reduced by mortality resulting from insects and maybe other episodic disturbances. We cannot judge if the model is realistic until the episodic disturbances are applied.

Tree mortality could be a problem, especially for smaller trees, as there is no suppression mortality. Episodic mortality from insects will maintain reasonable basal areas in the stand, however insects target larger trees, so the amount of suppressed smaller trees dying is likely underestimated.

The regeneration model could be improved. Currently it only regenerates species that were in the previous stand; however, in reality other species could become established as well. The regeneration model could be updated to regenerate additional species that were not in the stand at the time of harvest, according to the PAG.

The need for snag creation will most likely be reduced as a result of stage three in the ARWFSP. As episodic disturbances occur, there will be more snags and DWD, thus, fewer snags will need to be created in order to meet the snag target level.

## 7.1.3 RLS-PATH Based Model with a Look-ahead

An RLS-PATH based model with a look-ahead period was built for generating prescriptions for the ARWFSP. The goal was to build a model, which runs quickly and is able to be incorporated into the landscape simulator. The strength of the PATH algorithm is that it requires less storage and fewer computations than dynamic programming. One of the weaknesses is that the prescription leading to the optimum objective function might not be found. The probability of this occurring is greater when the benefits of an action are not realized within the next period. Using a look-ahead period reduces this risk and is therefore used in PREMO. The RLS increases the quickness of the model, by searching through a limited combination of harvest prescriptions. This is also the limitation of this strategy, because not all combinations are examined, therefore the best prescription could be missed. Issues to be further investigated include further reducing the time needed to solve these simulations, investigating the choice of variables and scalar measures for individual goals, consideration of the regeneration, adding prescribed burning, and updating costs, revenues, and flame length changes.

Currently, the prescription generation for the stand in the analysis takes approximately from 22 seconds for goal emphasis #2 to 41 seconds for goal emphasis #6. Effort has been made to reduce the computation time of a simulation, however, there are still several possibilities for decreasing the computation time in the model. Eliminating unnecessary calculations is an important step in reducing the calculation time. Another way to reduce the computational burden is to reduce the number and value of variables. This could be done by decreasing the look-ahead period, increasing the node interval, and reducing the number of species and diameter classes. Decreasing the look-ahead period often results in the harvest of smaller trees that might have otherwise been kept, but did not reach the large tree diameter by the end of the look-ahead period. Increasing the node interval reduces the number of prescriptions evaluated and therefore also the chance of finding better solutions. Reducing the number of species and diameter classes would not only result in a less realistic representation of the stand, but would also create a simpler stand structure.

One shortcoming of this model is that if the cutting from more than one diameter class is needed to improve the objective function, the algorithm decides that the objective function cannot be improved and continues to the next period. This problem is evident in the FBI. If there are two species groups with significant TPA in the lower diameter class, torching is possible. Harvesting the trees from only one of these classes will not reduce the potential for torching, but removing the trees from both classes will. Since the RLS-PATH algorithm only searches class by class, this improvement will be skipped. In the application by Yoshimoto et al. (1990), there were fewer species and larger diameter classes so this difficulty was not as significant. Future work could focus on creating groups of diameter and species classes. These groupings would solve the problem with finding harvest from several classes which improve the solution, as well as reduce solution time. Possible groupings are by diameter, relative height, or species.

Expanding the variables included in the RLS-PATH section could make a significant improvement to this model. As mentioned earlier, the consideration of snag recruitment within RLS-PATH could improve the achievement of goals. Furthermore, the regeneration is also not included in the RLS. This affects the ability to meet goals. In the case of the snags, the model currently searches through the tree list in order of the records until it finds a tree that meets the diameter requirement for a snag. For the sample analysis I had to restrict the model from creating snags of trees with a DBH ≥ 30", because PREMO was creating snags from the trees that were being reserved to meet the large tree goal. Therefore, it was not possible to meet both goals over time. If the trees from which to make snags could be optimized, then all diameters could be considered.

Regeneration greatly affects the FBI and the FEI, and by not including them in the RLS the possibility of meeting this goal in every period is reduced. In the RLS section a harvest prescription is evaluated at the present time and at the end of the look-ahead period. In the sample analysis this corresponds to evaluating the ability of a prescription to meet the goal measures of the emphasis at a given period and 50 years from that period. A harvest prescription is chosen without evaluating the effects of regeneration as a result of that harvest. Therefore, sometimes a harvest prescription is chosen which reduces the FBI and the FEI, however, once implemented, regeneration occurs, which increases the FEI, and also then torching might become possible, which increases the FBI.

PREMO does not include a simulation of prescribed burning. This needs to be added as the project progresses. Refinements also need to be made to the costs and revenues so that they reflect prices in the Applegate River Watershed. Finally, the process for changing flame length over time needs to be further investigated.

Initial sensitivity analysis was performed in order to determine the weights for individual goal measures within a simulation. However, choosing the weights for each goal measure is a fine art and simultaneous achievement of goals could be improved by refining their values.

## 7.2 Analysis

Another part of the project was to investigate the model's ability to meet the possible goals over time. In order to accomplish this, I applied the model to a sample analysis and assessed its performance.

### 7.2.1 Ability to Meet Individual Goals

The individual goal emphases were generally successful in meeting targets of their respective measures. The insect hazard goal reduction goal is achievable in every period after the harvest in the first period, and the large tree goal for enhancing fish habitat is attainable in every period after the period in which it is possible for large trees to exist for that stand.

In the stand of the sample analysis, the fire hazard goal was not achievable in every period. This is most often a result of the regeneration increasing the possibility of torching and increasing the amount of basal area that would burn in a fire. As mentioned earlier, changing the evaluation of the effect of regeneration could improve the ability to meet this goal.

For the most part, PREMO is able to create both simple and complex vertical structure in this stand after the initial period. However, it is difficult to maintain the number of snags per acre and the desired vertical structure at the same time. As mentioned earlier, one way of attaining complex structure is to reduce the number of trees per acre. Consequently, this could limit the presence of trees from which to create snags. Optimization of snags might help this problem. As can be seen in the goal emphases that do not focus on wildlife, the snag goal is not attainable through periodic mortality alone. Finally, the DWD goal is never achieved. PREMO simply monitors the amount of dead wood; it does not create any. The only way to currently meet the DWD goal would be to create snags far in excess of the goal and wait for them to fall and become DWD. Specific target levels for DWD could be scaled for each PAG, so that drier sites would not have as high of a goal as wetter sites, since historically these had less DWD.

### 7.2.2 Multi-Goal Analysis

The last three goal emphases considered multiple goals. The target levels of these goals were met in many of the periods and a positive PNV was produced. The ninth goal emphasis considered all goals. Based on the analysis, it appears feasible to get close to the goals of reducing risk of fire hazard and insect hazard, and enhancing wildlife and fish habitat, while producing a positive present net value.

The most competitive goals were having a target of five large trees and five large snags per acre in the stand. When the snags were created from any tree ≥ 30" DBH, the number of large trees per acre declined to zero over the planning horizon, as they were turned into snags. Therefore, snag creation was restricted to the mid-diameter range. The result of this was a creation of a two-cohort stand as there were several large trees per acre, but then no mid-range diameter trees, as they were all turned into snags. It is possible that five snags per acre will be too high of a goal for this PAG.

Another potential competition among goals can be found between the PNV and the ability to meet fire and insect risk goals. The ability to limit fire and insect hazard was reduced when harvests of few cubic feet per acre were not allowed. However, this minimum harvest volume requirement is important for achieving the goal of maintaining a positive PNV.

The PNV goal was more complimentary to the fire and insect goals than the wildlife and fish habitat goals. Goal emphasis #8, which focuses on habitat, produces a lower PNV than does goal emphasis #7. Stand basal area is not maintained below the insect hazard threshold in every period for goal emphases #7 and #9. This is also due to the minimum harvest requirement.

## 7.3 Summary of PREMO in the AWRFSP

The purpose of PREMO was to find prescriptions that would largely meet the goals for the stands. The accomplishment of this objective by PREMO demonstrates the power of its methodology. The project scientists originally thought that the goals would be more conflicting than the results indicate.

PREMO will be called from stages one and four of the landscape simulator. From the sample analysis, it appears that PREMO will be able to generate prescriptions that will generally meet the goals of the simulation. These goals include reducing the fire and insect hazard, enhancing wildlife and fish habitat, and providing a positive PNV.

## REFERENCES

Agee, J., and E. Goheen. 1998. Wind, insect, and disease inputs to the ARWFSP Model, version 2.3. Unpublished.

Bierman, H. Jr., and S. Schmitt. 1984. The Capital Budgeting Decision, Economic Analysis of Investment Projects. Sixth Edition. Macmillan Press. 545 p.

Brodie, J.D., D.M. Adams and C. Kao. 1978. Analysis of economic impacts on thinning and rotation for Douglas-fir using dynamic programming. For. Sci. 24:513-522.

Brodie, J.D., and C. Kao. 1979. Optimizing thinning Douglas-fir with three-descriptor dynamic programming to account for accelerated diameter growth. For. Sci. 25:655-672.

Cline, S.P., A.B. Berg, and H.M. Wight. 1980. Snag characteristics and dynamics in Douglas-fir forests, western Oregon. J. Wildl. Manage. 44(4):773-786.

Cousar, P.K., J. Sessions and K.N. Johnson. 1996. Methodology for estimating stand projections under different goals *in* Sierra Nevada Ecosystem Project: Final Report to Congress. Addendum. Davis: University of California, Centers for Water and Wildland Resources.

Davis, L.S. and K.N. Johnson. 1987. Forest Management. Third Edition. Mc-Graw Hill. 790p.

Dixon, Gary and Ralph Johnson. 1995. The Klamath Mountains geographic variant of the Forest Vegetation Simulator Version 6.1. USDA Forest Service, Washington Office, Forest Management Service Center, Fort Collins, CO. 19p. FMSC Internal Report.

Dubrasich, M.E., D.W. Hann, and J.C. Tappeiner II. 1997. Methods for evaluating crown area profiles of forest stands. Can. J. For. Res. 27:385-392.

Dykstra, D.P. 1984. Mathematical programming for natural resource management. McGraw-Hill, Inc.: New York. 318p.

Foster, B.B. 1979. Adjusting discount rates for risk. Journal of For. 77(5):287-288.

Goheen, D. 1998. Forest Service insect and disease. *in* Landscape Forestry -- With a Slant on Forest Health and the Urban Interface. A symposium by the Siskiyou Chapter Society of American Foresters and Southern Oregon University. November 13, 1998, Medford, Or.

Haight, R.G., J.D. Brodie, and D.M. Adams. 1985. Optimizing the sequence of diameter distributions and selection harvests for uneven-aged stand management. For. Sci. 31:451-462.

Haight, R.G., J.D. Brodie, and W.G. Dahms. 1985. A dynamic programming algorithm for optimization of lodgepole pine management. For. Sci. 31:321-330.

Haight, R.G., R.A. Monserud, and J.D. Chew. 1992. Optimal harvesting with stand density targets: managing rocky mountain conifer stands for multiple forest outputs. For. Sci. 38:554-573.

Harr, R.D., R.L. Fredriksen, and J. Rothacher. 1979. Changes in streamflow following timber harvest in southwestern Oregon. USDA Forest Service. Res. Pap. PNW-249. 22pp.

Hooke, R., and T.A. Jeeves. 1961. "Direct search" solution of numerical and statistical problems. Journal of the ACM, 8:212-229.

Kao, C., and J.D. Brodie. 1979. Determination of optimal thinning entry interval using dynamic programming. For. Sci. 25:672-674.

Kayser, J., H.L. Wedin, and J. Agee. 1998. Naturally regenerating Applegate's forests. Unpublished.

Maser, C., R.G. Anderson, K. Cromack, Jr., J.T. Williams, and R.E. Martin. 1979. Dead and down woody material. *in* Wildlife habitats in managed forests: the Blue Mountains of Oregon and Washington, Thomas, J.W., tech ed. Agric. Handb. 553. Washington, D.C., U.S. Dept. Agriculture. p. 78-95.

Mellen, K., and A. Ager. 1998. Coarse Woody Debris Model - Version 1.2. USDA Forest Service, Mt. Hood and Gifford Pinchot National Forest.

Ohmann, J.L., W.C. McComb and A.A. Zumrawi. 1994. Snag abundance for primary cavity-nesting birds on nonfederal forest lands in Oregon and Washington. Wildlife Society Bulletin 22:607-620.

Paredes V, G.L., and J.D. Brodie. 1987. Efficient specification and solution of the even-aged rotation and thinning problem. For Sci. 33:14-29.

Reiter, M.L., and R.L. Beschta. 1995. Effects of forest practices on water. Pp:1-185 *in* Cumulative Effects of Forest Practices in Oregon: Literature and Synthesis, Beschta, R.L., J.R. Boyle, C.C. Chambers, W.P. Gibson, S.V. Gregory, J. Grizzel, J.C. Hagar, J.L. Li, W.C., McComb, T.W. Parzybok, M.L. Reiter, G.H. Taylor and J.E. Warila. Oregon Department of Forestry, Salem, OR.

USDA Forest Service and USDI Bureau of Land Management. 1994. Record of Decision for Amendments to Forest Service and Bureau of Land Management Planning Documents Within the Range of the Northern Spotted Owl; Standards and Guidelines for Management of Habitat for Late-Successional and Old-Growth Forest Related Species Within the Range of the Northern Spotted Owl. Washington, DC: U. S. Government Printing Office.

USDI Bureau of Land Management, Medford District, USDA Forest Service, Rogue River National Forest, USDA Forest Service, Siskiyou National Forest, USDA Forest Service, PNW Research Station. 1994. Applegate Adaptive Management Area Ecosystem Health Assessment. Pp. 1-76.

Walters, D.K., D.W. Hann, and M.A. Clyde. 1985. Equations and tables for predicting gross total stem volumes in cubic feet for six major conifers of Southwest Oregon. Forest Research Laboratory, Oregon State University. Research Bulletin 50. 36p.

Yoshimoto, A., J.D. Brodie, and J. Sessions. 1994. A new heuristic to solve spatially constrained long-term harvest scheduling problems. For. Sci. 40:365-396.

Yoshimoto, A., R.G. Haight, and J.D. Brodie. 1990. A comparison of the pattern search algorithm and the modified PATH algorithm for optimizing an individual tree model. For. Sci. 36:394-412.

Yoshimoto, A., G.L. Paredes V, and J.D. Brodie. 1988. Efficient optimization of an individual tree growth model USDA For. Serv. Gen. Tech. Rep. RM-161: 154-162.

## APPENDICES

# Appendix A: Growth Subroutines

The code for PREMO is presented in this Appendix. It is written in the programming language C++. It is divided into several functions that are called from MAIN. The programs for diameter growth include GROWT, BRATIO, and HTDBH. The programs for height growth are GROWT, ECOCLASS, HTCALC, HTDBH, HTGR5, and SICHG. The program for crown ratio change is GROWT and the calculations for crown width are in main. The subroutines for calculating volume were also translated and they are BFVOL, FORMCL, and RXDIB.

**MAIN**

```
#include "var.h"
int fix=0;
extern float abirth[linesor], ag;
float oldabirth[linesor];
//double saveht[linesor],
double k;
float elev=45, slope=5, aspect=90;
double snagscost, snags, nusnags, snagsmorttemp, snagtrees;

// taper
#include "windows.h"
#define TAPER_DLL
#include "taper.h"
HINSTANCE   DLL;              // handle to the dll
diDIB       pfdiDIB;          // pointer to function in dll
//MerchHeight pfMerchHeight;     // pointer to function in dll

// goal combinations
float gc[goalcombos];
int ksnag, kstree, kdwd, createsnags, mcflimit;
int gcctr, origlines, a, b, c, zerosflag, lastctr;
float fishtreesgoal, ccgoal;
double verticalvargoal, snagsgoal, snagsgoaltemp, dwdgoal, fbigoal, feigoal;
float ftg[goalcombos]={5,5,5,5,5,5,5,5,5},
ccg[goalcombos]={60,60,60,60,80,60,60,60,60};
double vertg[goalcombos]={50,50,50,50,100,50,50,50,50};
double sg[goalcombos]={5,5,5,5,5,5,5,5,5};
double dwdg[goalcombos]={120,120,120,120,120,120,120,120,120};
double fbig[goalcombos]={30,30,30,30,30,30,30,30,30};
double feig[goalcombos]={30,30,30,30,30,30,30,30,30};
double origcr[linesor], origdwdpa[linesor], origdwdld[linesor],origdwdln[linesor];
double origdwddensity[linesor], origmorttpa[linesor], dwdjunk;
double origtpa[linesor], origdbh[linesor], oright[linesor], origsdensity[linesor];
int origld[linesor], origsp[linesor], origreportsp[linesor], origsnagage[linesor];
int origdwdage[linesor], origdwdflag[linesor], dwdgrowsp, dwdreportsp;
int origdwdsp[linesor];
double totalmcfharvest;
double tinsectdev=0, tfishtreesdev=0, tccdev=0,tfei=0,tfbi=0,tverticaldev=0, tsnagsdev=0,
tdwddev=0;

// variables for mortality
double morttot, lpinemort, mpinemort, lpinekill, mpinekill, lpinetpa, mpinetpa;
int htorder[linesor];
double plotba, windmortba, mortba, windmorttpa, talltrees, heighttall5trees;
int snagflag[linesor][periods], dwdage[linesor][periods][periods],
snagage[linesor][periods];
int dwdflag[linesor][periods][periods][2], snagsp[linesor][periods];
int dwdsp[linesor][periods][periods];
double snagtpa[linesor][periods], snaght[linesor][periods];
double snagdensity[linesor][periods], snagdbh[linesor][periods];
double dwdln[linesor][periods][periods][2], dwdld[linesor][periods][periods][2];
double dwdpa[linesor][periods][periods][2], dwddensity[linesor][periods][periods];
double reportsnag[5][5], reportdwd[2][5][5], createsnagsmorttpa[linesor];
int dctr, dper, actr, lctr;

//regen
double harvesttrees, totaltrees;
int regenper, rsavelines, rsavesp[linesor];
double rsavetpa[linesor], rsaveba[linesor],rsavemaxhrvtpa[linesor];

// external variables
int lines, oldlines;
double pba, futba[linesor];
double morttpa[linesor];
double flameln, oldflameln;

// variables for output
FILE *growfp;
int flagforperiod=1, flagforhtlc=1;
FILE *editfp, *hsfp, *davidfp;
double mcfvolharvest, baharvest, reportnusnags;

// variables for inputing data
int plot[linesor], treeid[linesor];
double cr[linesor] ;
char *spec, *spec2;
double tpa[linesor], dbh[linesor], ht[linesor], sdensity[linesor];
int ld[linesor], sp[linesor], reportsp[linesor];
int ctr, fscanfint, dwdctr, dwdlines, deadlines;
FILE *ofp;

// variables for functions
```

```
int vegclass;

// variables for standard deviation
int sdctr=0;
double sumobj, meanobj, sqterm, sumsqterm, stdev;
int minflag, ctrformaxhrvtemp, editfpctr;

// variables for saving maximum objective function
int maxctr, maxptr[linesor];
double maxobj[linesor], minsumdev[linesor];
double maxhrvtpa[linesor], maxhrvtpatemp;
double goalattainment[periods],presentnetvalue[periods];
double totalvalue, totalsumdev;

// variables for stabilizing the objective function
//float
double stblhrvtpa[linesor];
double lastdev, lastobj;
int stblctr, stblflag;

// variables for growth and value calculations
double  rescr[linesor];
double hrvtpa[linesor], restpa[linesor], resdbh[linesor], resht[linesor];
double objtpa[linesor], objdbh[linesor];
double vol[linesor], resvol[linesor];
double obj, presentobj, futureobj, saveobj;
int fixresctr, solctr, solflag, hrvflag;
double ba[linesor], resba[linesor];
double x[linesor];
int first, last, baptr[linesor], initialize=1;
double htg[linesor],dg[linesor];
double cw[linesor], rescw[linesor];
double
cwco1[totsp]={2.4922,4.4215,4.0920,2.8541,7.5183,2.8541,3.1146,3.2367,7.5183,3.8166}
;
double
cwco2[totsp]={0.8544,0.5329,0.4912,0.6400,0.4461,0.6400,0.5780,0.6247,0.4461,0.5229}
;
double
cwco3[totsp]={0.1400,0.5170,0.4120,0.4070,0.8150,0.4070,0.3450,0.4060,0.8150,0.4520}
;
double presentsumdev, futuresumdev, sumdev, savesumdev;
double olddbh[linesor], oldht[linesor], oldcr[linesor], growctr;
double presentpba;
double bfv[linesor], bfrev[linesor];

// variables for multiple periods
int per;
char goalemphasisname[2];
```

```
// species characters
char bochar[3]="BO", dfchar[3]="DF", icchar[3]="IC", kpchar[3]="KP";
char mchar[3]="M", ppchar[3]="PP", rfchar[3]="RF", spchar[3]="SP";
char tochar[3]="TO", wfchar[3]="WF";

/******************************** IMPORTANT
************************************************/
//#define DAVID_RUN
#define HEIDI_RUN

#ifdef DAVID_RUN
    void main(int argc, char *argv[] ) // David
#endif

#ifdef HEIDI_RUN
    void main(void)
#endif
/**********************************************************************
*******************/


// beginning of main
{
        //SOME LOCAL VARIABLES USED BY HEIDI AND DAVID
int GoalEmp, CurPer, TreeList;
FILE *index;
int ScanStatus,IndexNo,Found;
char TreeFileName[100]="";
char Temp[100]="";



#ifdef DAVID_RUN                                //NEEDS to be defined when running
on David's computer
/* David G. added the following to allow the use of a .bat file to allow the program to be
called up
as a separate executable from the main Applesite program.  The Applesite program will
create the .bat
file each time this Standopt needs to be run.  For any run, the input arguments passed will
be:

  -- The name of this .exe (standopt.exe which goes into argv[0])
  -- Which goal # emphasis to use ( a default value of 6 will be made by Applesite but will
change as needed)
  -- What the current period is
  -- What tree list to use...this will be a index number that can be cross referenced in the file
called
     TREEINDEX.txt which will be kept in the \Constant\ directory.
*/
```

```
    GoalEmp = atoi(argv[1]);
    CurPer = atoi(argv[2]);
    TreeList = atoi(argv[3]);

    if(TreeList == 209)
    {
        printf("\nReceived a Treelist value of 209, which is water,barren, G/F, or shrub -
not optimizing!\n");
        exit(1);
    }



    char TreeIndexFile[90]= "G:\\applesite\\inputs\\Constant\\Mastertreeindex.txt";
    char GrowoutDir[90]="g:\\applesite\\outputs\\Standopt\\growout_";
    char HtoutDir[90]="g:\\applesite\\outputs\\Standopt\\ht_";
    char HsoutDir[90]="g:\\applesite\\outputs\\Standopt\\hs_";
    char DoutDIR[90]="g:\\applesite\\outputs\\Standopt\\david_";

#ifdef DEBUG_ON
    printf("Goal Emphasis value is:  %d\n",GoalEmp);
    printf("Current Period is:  %d\n",CurPer);
    printf("Tree List index number is:  %d\n", TreeList);
    printf("\ntreeindexfile=%s\n",TreeIndexFile);
#endif

    Temp[0] = '\0';
    strcat(Temp, GrowoutDir);
    strcat(Temp, argv[3]);
    strcat(Temp, ".txt");
    growfp = fopen(Temp,"w");

    Temp[0] = '\0';
    strcat(Temp, HtoutDir);
    strcat(Temp, argv[3]);
    strcat(Temp, ".txt");
    editfp=fopen(Temp,"w");

    Temp[0] = '\0';
    strcat(Temp, HsoutDir);
    strcat(Temp, argv[3]);
    strcat(Temp, ".txt");
    hsfp=fopen(Temp,"w");

    Temp[0] = '\0';
    strcat(Temp, DoutDir);
    strcat(Temp, goalemphasisname);
    strcat(Temp, ".txt");
    davidfp=fopen(Temp,"w");
#endif
```

```
#ifdef HEIDI_RUN                              //NEEDS to be defined when running on
Heidi's computer
// Heidichange

    GoalEmp=8;
    CurPer=0;
    TreeList=11;// 11 or 12

    char goalemphasisname[2];
    goalemphasisname[0]='8';
    goalemphasisname[1]='\0';


    char GrowoutDir[90]="c:\\apple\\vegtype\\output\\growout_";
    char HtoutDir[90]="c:\\apple\\vegtype\\output\\ht_";
    char HsoutDir[90]="c:\\apple\\vegtype\\output\\hs_";
    char TreeIndexFile[90]="C:\\apple\\vegtype\\input\\Mastertreeindex.txt";
    char DoutDir[90]="c:\\apple\\vegtype\\output\\david_";

    Temp[0] = '\0';
    strcat(Temp, GrowoutDir);
    strcat(Temp, goalemphasisname);
    strcat(Temp, ".txt");
    growfp = fopen(Temp,"w");

    Temp[0] = '\0';
    strcat(Temp, HtoutDir);
    strcat(Temp, goalemphasisname);
    strcat(Temp, ".txt");
    editfp=fopen(Temp,"w");

    Temp[0] = '\0';
    strcat(Temp, HsoutDir);
    strcat(Temp, goalemphasisname);
    strcat(Temp, ".txt");
    hsfp=fopen(Temp,"w");

    Temp[0] = '\0';
    strcat(Temp, DoutDir);
    strcat(Temp, goalemphasisname);
    strcat(Temp, ".txt");
    davidfp=fopen(Temp,"w");

#endif
```

```
// ********************* This is really the beginning of the actual programming stuff
*****************


    index = fopen(TreeIndexFile,"r");

    if (index == NULL)
        fprintf(stderr, "opening of treeindex.txt failed:  %s\n",  strerror(errno));
    else
    {
        #ifdef DEBUG_ON
                printf("File: INDEX.txt opened with no problems in mode READ!\n");
        #endif
    }

    //now start searching through the file till you find the TreeList variable and set the
FileName string
    //to the next parameter in the index.txt file which will be opened later by this program
as the
    //input treelist.

    Found = 0;
    while ((ScanStatus=fscanf(index,"%d %s",&IndexNo,TreeFileName))!=EOF)
    {
        if (IndexNo == TreeList)
        {
            printf("I found the correct file for tree list %d
.....%s\n",IndexNo,TreeFileName);
            Found = 1;
            break;
        }
    }
    if (Found == 0)
    {
        printf("There appears to be no treelist available for IndexNo: %d - Bailing
out!\n",IndexNo);
        exit(0);
    }


    fclose(index);

//========================== End of what David has added so far
================================


    // flame length increase
    double treestimesdbh, hrvtreestimesdbh,preavgdbh,hrvavgdbh,dtodratio;

    // other
    double standhtlc;
    int a,b,c, snagdclass, state, stdevctr;//, snaglow, snaghigh, snaghid;
    int introsnagage[3][5]={5,15,30,50,70,
                            5,10,15,30,40,
                            0,5,10,20,25};
    double introsnagden[3][5]={0.416,0.350,0.271,0.193,0.138,
                               0.383,0.324,0.276,0.168,0.121,
                               0.452,0.347,0.266,0.157,0.120};
    int introdwdage[3][5]={5,15,40,85,130,
                           5,15,35,70,105,
                           5,15,20,40,55};
    double introdwdden[3][5]={0.426,0.378,0.280,0.195,0.136,
                              0.419,0.361,0.274,0.193,0.136,
                              0.397,0.349,0.269,0.180,0.133};

//  diDIB      pfdiDIB;        // pointer to function in dll



// initialize

    for (ctr=0; ctr<linesor; ctr++)
    {  plot[ctr]=0;
       spec="";
       tpa[ctr]=0;
       dbh[ctr]=0;
       ht[ctr]=0;
       cr[ctr]=0;
       maxhrvtpa[ctr]=0;
       minsumdev[ctr]=1.7*pow(10,308);
       sdensity[ctr]=0;
       createsnagsmorttpa[ctr]=0;
    }


// input data
*****************************************************************************
***

    Temp[0] = '\0';                                 //David added these
    strcat(Temp, GrowoutDir);
//  strcat(Temp, argv[3]);                          // David
    strcat(Temp, goalemphasisname);
    strcat(Temp, ".txt");
    growfp = fopen(Temp,"w");


    Temp[0] = '\0';                                 //David added these
    strcat(Temp, HtoutDir);
//  strcat(Temp, argv[3]);                          // David
```

```
    strcat(Temp, goalemphasisname);
    strcat(Temp, ".txt");
    editfp=fopen(Temp,"w");

    Temp[0] = '\0';                                         //David added these
    strcat(Temp, HsoutDir);
//  strcat(Temp, argv[3]);                                  // David
    strcat(Temp, goalemphasisname);
    strcat(Temp, ".txt");
    hsfp=fopen(Temp,"w");




    //ofp=fopen("c:\\apple\\vegtype\\p0960.txt","r");
    ofp=fopen(TreeFileName,"r");                            //David did this to
comply with .bat stuff
    if (ofp == NULL)
        fprintf(stderr, "opening of the treelist %s failed:  %s\n", TreeFileName,
strerror(errno));
    else
    {
        #ifdef DEBUG_ON
                printf("File: %s opened with no problems in mode
READ!\n",TreeFileName);
        #endif
    }

//  ofp=fopen("c:\\apple\\vegtype\\p0960.txt","r");

    ctr=0;
    dwdctr=0;

    while ((fscanfint=fscanf(ofp,"%d %d",&plot[ctr],&ld[ctr]))!=EOF)
    {
        if (ld[ctr]==0 || ld[ctr]==1)
        {
            fscanf(ofp,"%lf %d %d %lf %lf
%lf",&tpa[ctr],&sp[ctr],&reportsp[ctr],&dbh[ctr],
                &ht[ctr],&cr[ctr]);

            origdbh[ctr]=dbh[ctr];
            if (dbh[ctr]==0)
            {
                dbh[ctr]=0.1;
                origdbh[ctr]=0.1;
            }
            oright[ctr]=ht[ctr];
            origcr[ctr]=cr[ctr];
            origsp[ctr]=sp[ctr];
            origreportsp[ctr]=reportsp[ctr];

            if (ld[ctr]==0)
            {
                fscanf(ofp,"%d",&state);
                morttpa[ctr]=tpa[ctr];
                tpa[ctr]=0;
                if (dbh[ctr]<15)
                    snagdclass=2;
                else if (dbh[ctr]<25)
                    snagdclass=1;
                else
                    snagdclass=0;

                if (state==0)
                    state=1;
                else
                    state--;

                snagage[ctr][0]=introsnagage[snagdclass][state];
                sdensity[ctr]=introsnagden[snagdclass][state];
                origsnagage[ctr]=introsnagage[snagdclass][state];
                origsdensity[ctr]=sdensity[ctr];
            }

            origtpa[ctr]=tpa[ctr];
            origmorttpa[ctr]=morttpa[ctr];
            origld[ctr]=ld[ctr];

            restpa[ctr]=tpa[ctr];
            ba[ctr]=pow((dbh[ctr]/24),2)*pi;
            baptr[ctr]=ctr;

            if (ht[ctr]>4.5)
                cw[ctr]=cwco1[sp[ctr]]*pow(dbh[ctr],cwco2[sp[ctr]]);
            else
                cw[ctr]=cwco3[sp[ctr]]*ht[ctr];

            ctr++;
        }
        else
        {
            fscanf(ofp,"%lf %d %d %lf %lf %lf %d",&dwdpa[dwdctr][0][0][0],
&dwdsp[dwdctr][0][0],
                &dwdreportsp, &dwdld[dwdctr][0][0][0],
                &dwdln[dwdctr][0][0][0], &dwdjunk, &state);

            if (dwdld[dwdctr][0][0][0]<15)
                snagdclass=2;
            else if (dwdld[dwdctr][0][0][0]<25)
```

```
                snagdclass=1;
            else
                snagdclass=0;

            if (state==0)
                state=1;
            else
                state--;

            origdwdpa[dwdctr]=dwdpa[dwdctr][0][0][0];
            origdwdld[dwdctr]=dwdld[dwdctr][0][0][0];
            origdwdln[dwdctr]=dwdln[dwdctr][0][0][0];
            origdwdsp[dwdctr]=dwdsp[dwdctr][0][0];
            dwddensity[dwdctr][0][0]=introdwdden[snagdclass][state];
            origdwddensity[dwdctr]=introdwdden[snagdclass][state];
            dwdage[dwdctr][0][0]=introdwdage[snagdclass][state];
            origdwdage[dwdctr]=introdwdage[snagdclass][state];
            origdwdflag[dwdctr]=1;
            dwdflag[dwdctr][0][0][0]=1;

            dwdctr++;
        }
    }

    lines=ctr;
    origlines=lines;
    dwdlines=dwdctr;
    if (dwdlines>lines)
        deadlines=dwdlines;
    else
        deadlines=lines;

    fprintf(growfp,"Plot Species TPA   DBH    HT   CR   CW  at beginning of period
%d yr=%d\n",
        per,per*5+1998);
    for (ctr=0; ctr<lines; ctr++)
    {
        fprintf(growfp,"%4.0d%6d%9.2f%7.2f%7.1f%4.0f%7.2f       ctr=%d\n",
            plot[ctr],sp[ctr],tpa[ctr],dbh[ctr],ht[ctr],cr[ctr],cw[ctr],ctr);
    }

    vegclass=vegclassification(dbh,tpa,cw,reportsp,ba);
    fprintf(davidfp,"Period\n%d %d %d\n",per,lines,vegclass);
    for (ctr=0; ctr<lines; ctr++)
        fprintf(davidfp,"%d %f %.2f %.1f %.0f %.2f\n",
            sp[ctr],tpa[ctr],dbh[ctr],ht[ctr],cr[ctr],cw[ctr]);

    fprintf(growfp,"Thank you\n");

    #ifdef DEBUG_ON
```

```
#ifdef DAVID_RUN
    //create a check index file to put the treelist value and vegclass value into for checking
    //after all prescriptions are run - will be checked by the Applesite program

FILE *CheckIndex;
char CheckFile[60] = "G:\\applesite\\outputs\\STANDOPT\\CheckIndex.txt";

    CheckIndex = fopen(CheckFile,"a");

    fprintf(CheckIndex,"%d\t%d\n",TreeList,vegclass);
    fclose(CheckIndex);

    exit(1);
#endif
#endif


/* This section is to check the growth model */

    if (fix==1)
    {
        for (per=0; per<2/*periods*/; per++)
        {
            initialize=1;

    standhtlc=sumdevfn(sp,ba,pba,cw,dbh,restpa,morttpa,0,presentobj,ht,cr,flameln,editfp)
;
            presentpba=scalef(12.8,41.8);
            presentpba=formcl(sp[0],dbh[0]);
            presentpba=objfunction(tpa,dbh,vol);
            vegclass=vegclassification(dbh,tpa,cw,reportsp,ba);
            presentpba=standhtlcfn(ht,tpa,cr,flameln,0,1);
    //      regen(tpa,ba,sp,maxhrvtpa,lines);
            volharvest(tpa,dbh,ht);
//          if (pag==5 || pag==6 || pag==7)
//              rootdisease(vol,sp);
            standhtlc=htdbh(sp[0],dbh[0],ht[0],0);
            standhtlc=htgr5(sp[0],85,ba[0],0.1,cr[0],ht[0]);
            growtrees(tpa,dbh,ht);
            vols(tpa, dbh, dg, sp, ht);
            standhtlc=verticalcomplexity(ht,cr,tpa);
            flameln=flamelnfn(dbh,tpa,cw,reportsp,ba);
            standhtlc=standhtlcfn(resht,tpa,rescr,flameln,0,1);

            for (ctr=0; ctr<lines; ctr++)
                fprintf(growfp,"vol=%4.2f\n",bfv[ctr]);
        //      fprintf(growfp,"htg=%4.2f dg=%4.2f
vol=%4.2f\n",htg[ctr],dg[ctr],bfv[ctr]);
            fprintf(growfp,"Plot Species TPA   DBH    HT  CR  CW  at beginning of
period %d yr=%d\n",
                per+1,(per+1)*5+1998);
            for (ctr=0; ctr<lines; ctr++)
            {   fprintf(growfp,"%4.0d%6d%9.2f%7.3f%7.1f%5.1f%6.2f  ctr=%d\n",

    plot[ctr],sp[ctr],tpa[ctr],resdbh[ctr],resht[ctr],rescr[ctr],rescw[ctr],ctr);
                dbh[ctr]=resdbh[ctr];
                ht[ctr]=resht[ctr];
                cr[ctr]=rescr[ctr];
                cw[ctr]=rescw[ctr];
                ba[ctr]=futba[ctr];

            }
            fprintf(growfp,"htlc=%6.2f\n",standhtlc);

        }
    }

    else // no fix, actual run
    {

#ifdef DAVID_RUN
        DLL = LoadLibrary( "g:\\applesite\\standopt\\sourcefiles\\taper.dll" );
#endif

#ifdef HEIDI_RUN
        DLL = LoadLibrary("taper.dll");
#endif
        if( !DLL )
            printf( "couldn't load taper dll\n" );


        // set the pointers to the functions' address
        pfdiDIB        = (diDIB)GetProcAddress(DLL,"diDIB");

        // check to make sure you got it!
        if( pfdiDIB  )
        {
            fprintf( stdout, "found diDIB\n" );
        }

        for (gcctr=GoalEmp;gcctr<(GoalEmp + 1);gcctr++)          //David did this to
comply with .bat stuff
        {
            k=0.001;  // this is a multiplier for sumdev in objective
            printf("Goal Combination %d\n",gcctr);
            fishtreesgoal=ftg[gcctr];
            ccgoal=ccg[gcctr];
            verticalvargoal=vertg[gcctr];
            snagsgoal=sg[gcctr];
            dwdgoal=dwdg[gcctr];
```

```
fbigoal=fbig[gcctr];
feigoal=feig[gcctr];
lines=origlines;
tinsectdev=0;
tfishtreesdev=0;
tccdev=0;
tfei=0;
tfbi=0;
tverticaldev=0;
tsnagsdev=0;
tdwddev=0;

if (gcctr<8)
{
    for (ctr=0; ctr<goalcombos; ctr++)
        gc[ctr]=0;
    ksnag=0;
    kstree=0;
    kdwd=0;
}
else// all gcchange
{
    for (ctr=0; ctr<goalcombos; ctr++)
        gc[ctr]=1;
    ksnag=1;
    kstree=1;
    kdwd=1;
    mcflimit=1;
    createsnags=1;
    gc[4]=100;//000;
    gc[0]=100;//00000;
    gc[3]=100000;
}

if (gcctr==0)     // fire
{
    gc[0]=1;
    mcflimit=0;
    createsnags=0;
}
else if (gcctr==1)     // insects
{
    gc[3]=1;
    mcflimit=0;
    createsnags=0;
}
else if (gcctr==2)     // fish
{
    gc[4]=1;
    mcflimit=0;
```

```
        createsnags=0;
    }
    else if (gcctr==3 || gcctr==4)    // wildlife
    {
        gc[1]=1;
        gc[2]=1;
        ksnag=1;
        kstree=1;
        kdwd=1;
        mcflimit=0;
        createsnags=1;
    }
    else if (gcctr==5)    // PNV
    {
        mcflimit=1;
        createsnags=0;
    }
    else if (gcctr==6)    // fire, insects, PNV
    {
        gc[0]=100;
        gc[3]=1000;
        mcflimit=1;
        createsnags=0;
        k=0.0001;
    }
    else if (gcctr==7)    // fish, wildlife, PNV
    {
        gc[1]=1;
        gc[2]=1;
        ksnag=1;
        kstree=1;
        kdwd=1;
        gc[4]=1;
        mcflimit=1;
        createsnags=1;
    }

    for (ctr=0; ctr<linesor; ctr++)
    {
        sdensity[ctr]=0;
        tpa[ctr]=0;
        morttpa[ctr]=0;
        dbh[ctr]=0;
        ht[ctr]=0;
        cr[ctr]=0;
        ld[ctr]=4;
        sp[ctr]=0;
        for (a=0; a<periods; a++)
        {
            snagsp[ctr][a]=999;
            snagflag[ctr][a]=0;
            snagage[ctr][a]=0;
            snagtpa[ctr][a]=0;
            snaght[ctr][a]=0;
            snagdensity[ctr][a]=0;
            snagdbh[ctr][a]=0;

            for (b=0; b<periods; b++)
            {
                dwdsp[ctr][a][b]=999;
                dwdage[ctr][a][b]=0;
                dwddensity[ctr][a][b]=0;
                for (c=0; c<2; c++)
                {
                    dwdln[ctr][a][b][c]=0;
                    dwdld[ctr][a][b][c]=0;
                    dwdpa[ctr][a][b][c]=0;
                    dwdflag[ctr][a][b][c]=0;
                }
            }
        }
    }

    for (a=0; a<5; a++)
        for (b=0; b<5; b++)
        {
            reportsnag[a][b]=0;
            for (c=0; c<2; c++)
                reportdwd[c][a][b]=0;
        }

    for (ctr=0; ctr<lines; ctr++)
    {
        tpa[ctr]=origtpa[ctr];
        morttpa[ctr]=origmorttpa[ctr];
        ld[ctr]=origld[ctr];
        dbh[ctr]=origdbh[ctr];
        ht[ctr]=oright[ctr];
        cr[ctr]=origcr[ctr];
        sp[ctr]=origsp[ctr];
        sdensity[ctr]=origsdensity[ctr];
        snagage[ctr][0]=origsnagage[ctr];
        reportsp[ctr]=origreportsp[ctr];
        maxhrvtpa[ctr]=0;
        minsumdev[ctr]=1.7*pow(10,308);

        restpa[ctr]=tpa[ctr];
        ba[ctr]=pow((dbh[ctr]/24),2)*pi;
        baptr[ctr]=ctr;
        if (ht[ctr]>4.5)
```

```
            cw[ctr]=cwco1[sp[ctr]]*pow(dbh[ctr],cwco2[sp[ctr]]);
        else
            cw[ctr]=cwco3[sp[ctr]]*ht[ctr];

    }

    for (ctr=0; ctr<dwdlines; ctr++)
    {
        dwdpa[ctr][0][0][0]=origdwdpa[ctr];
        dwdld[ctr][0][0][0]=origdwdld[ctr];
        dwdln[ctr][0][0][0]=origdwdln[ctr];
        dwddensity[ctr][0][0]=origdwddensity[ctr];
        dwdflag[ctr][0][0][0]=origdwdflag[ctr];
        dwdage[ctr][0][0]=origdwdage[ctr];
        dwdsp[ctr][0][0]=origdwdsp[ctr];
    }

    flameln=flamelnfn(dbh,tpa,cw,reportsp,ba);

    for (per=0; per<periods; per++)
    {

        printf("period %d\n",per);
        if (per==14)
            printf("");


        hrvtpa[ctr]=0;
        hrvflag=0;
        maxctr=0;

        presentpba=0;
        for (fixresctr=0; fixresctr<lines; fixresctr++)
        {
            presentpba+=ba[fixresctr]*tpa[fixresctr];
            maxptr[fixresctr]=99999999;
        }
        solflag=1;


        do
        {


            if (maxctr==0)
            {
                lastdev=1.7*pow(10,308);
                lastobj=-99999999999;
            }
            else
```

```
          {
               lastobj=maxobj[maxctr-1];
               lastdev=minsumdev[maxctr-1];
          }

          zerosflag=0;
          for (ctr=0; ctr<lines; ctr++)
          {
               for (solctr=0; solctr<maxctr; solctr++) // check to see if this tree
               {                                       // has already been
optimized
                    if (ctr==maxptr[solctr])
                         solflag=0;
               }

               if (tpa[ctr]-hrvtpa[ctr]==0)
                    solflag=0;

               if (solflag==1)
               {
                    do
                    {
                         if (zerosflag==1)
                              hrvtpa[ctr]+=HARVESTNUMBER;

                         if (maxctr==24 && ctr==22)
                              printf("");

                         if(hrvtpa[ctr]>tpa[ctr] && hrvflag==0)
                         {
                              hrvtpa[ctr]=tpa[ctr];
                              hrvflag=1;
                         }

                         if(zerosflag==0)
                         {
                              for (fixresctr=0; fixresctr<lines; fixresctr++) //
initialize restpa
                                   restpa[fixresctr]=tpa[fixresctr]-hrvtpa[fixresctr];

                              for (fixresctr=0; fixresctr<lines; fixresctr++) //
initialize resba
                              {
                                   resba[fixresctr]=ba[ctr];
                                   if (restpa[fixresctr]==0)
                                        resba[fixresctr]=0;
                              }
                    //        oldflameln=flameln;
                         }
                         else
```

```
{
    restpa[ctr]=tpa[ctr]-hrvtpa[ctr];
    resba[ctr]=ba[ctr];
    if (restpa[ctr]==0)
        resba[ctr]=0;

    restpa[lastctr]=tpa[lastctr]-hrvtpa[lastctr];
    resba[lastctr]=ba[lastctr];
    if (restpa[lastctr]==0)
        resba[lastctr]=0;
}

if (solflag==1)
    printf("");

volharvest(hrvtpa,dbh,ht);
vols(hrvtpa, dbh, dg, sp, ht);
presentobj=objfunction(hrvtpa,dbh,vol);
presentsumdev=sumdevfn(sp,ba,presentpba,cw,dbh,restpa,
    morttpa,0,presentobj,ht,cr,flameln,editfp);

harvesttrees=0;
totaltrees=0;
treestimesdbh=0;
hrvtreestimesdbh=0;
for (fixresctr=0; fixresctr<lines; fixresctr++)
{
    harvesttrees+=hrvtpa[fixresctr];
    totaltrees+=restpa[fixresctr];
    treestimesdbh+=restpa[fixresctr]*resdbh[fixresctr];

hrvtreestimesdbh+=hrvtpa[fixresctr]*resdbh[fixresctr];

}

if (harvesttrees>0 && preavgdbh>0)
{
    preavgdbh=treestimesdbh/totaltrees;
    hrvavgdbh=hrvtreestimesdbh/harvesttrees;
    dtodratio=hrvavgdbh/preavgdbh;
}
else
    dtodratio=0;

if (dtodratio<0.9 && harvesttrees>0)
    flameln-=1;
if (harvesttrees==0)
    flameln+=0.33;
if (flameln<0.33)
    flameln=0.33;
```

```
                    else if (flameln>15)
                        flameln=15;

                    if (zerosflag==0)
                    {
                        for (fixresctr=0; fixresctr<lines; fixresctr++)
                        {
                            olddbh[fixresctr]=dbh[fixresctr];
                            oldht[fixresctr]=ht[fixresctr];
                            oldcr[fixresctr]=cr[fixresctr];
                            oldabirth[fixresctr]=abirth[fixresctr];
                        }
                        oldflameln=flameln;
                    }

                    for (growctr=0; growctr<growperiods; growctr++)
                    {
                        growtrees(restpa,dbh,ht);

                        for (fixresctr=0; fixresctr<lines; fixresctr++)
                        {
                            dbh[fixresctr]=resdbh[fixresctr];
                            ht[fixresctr]=resht[fixresctr];
                            cr[fixresctr]=rescr[fixresctr];
                        }

                    }

                    pba=0;
                    for (fixresctr=0; fixresctr<lines; fixresctr++)
                    {
                        pba+=futba[fixresctr]*restpa[fixresctr];

                        if (ht[fixresctr]>4.5)

cw[fixresctr]=cwco1[sp[fixresctr]]*pow(dbh[fixresctr],cwco2[sp[fixresctr]]);
                        else
                            cw[fixresctr]=cwco3[sp[fixresctr]]*ht[fixresctr];
                    }


                    vols(restpa, resdbh, dg, sp, resht);

futureobj=objfunction(restpa,resdbh,vol)/(pow((1+interestrate),(per+1)*5));
                    obj=presentobj+futureobj;


futuresumdev=sumdevfn(sp,futba,pba,rescw,resdbh,restpa,
                        morttpa,1,futureobj,resht,rescr,flameln,editfp);
                    sumdev=futuresumdev+presentsumdev;
                    for (fixresctr=0; fixresctr<lines; fixresctr++)
                    {
                         dbh[fixresctr]=olddbh[fixresctr];
                         ht[fixresctr]=oldht[fixresctr];
                         cr[fixresctr]=oldcr[fixresctr];
                         abirth[fixresctr]=oldabirth[fixresctr];
                         if (ht[fixresctr]>4.5)

cw[fixresctr]=cwco1[sp[fixresctr]]*pow(dbh[fixresctr],cwco2[sp[fixresctr]]);
                         else
                              cw[fixresctr]=cwco3[sp[fixresctr]]*ht[fixresctr];
                    }
                    flameln=oldflameln;

                    if (obj>minobjvalue)
                    {
                         if (gcctr>=5)
                         {
                              if ((sumdev-k*obj)<minsumdev[maxctr])
                              {
                                   minsumdev[maxctr]=sumdev-k*obj;
                                   maxobj[maxctr]=obj;
                                   maxptr[maxctr]=ctr;
                                   maxhrvtpatemp=hrvtpa[ctr];
                                   saveobj=presentobj;
                              //   for (fixresctr=0; fixresctr<lines; fixresctr++)
                              //        saveht[fixresctr]=resht[fixresctr];
                              }
                              else if ((sumdev-k*obj)==minsumdev[maxctr])
                              {
                                   if (obj>maxobj[maxctr])
                                   {
                                        maxobj[maxctr]=obj;
                                        maxptr[maxctr]=ctr;
                                        maxhrvtpatemp=hrvtpa[ctr];
                                        saveobj=presentobj;
                                   //   for (fixresctr=0; fixresctr<lines;
fixresctr++)

                                   //        saveht[fixresctr]=resht[fixresctr];
                                   }
                              }
                         }
                         else if (sumdev<minsumdev[maxctr])
                         {
                              minsumdev[maxctr]=sumdev;
                              maxobj[maxctr]=obj;
                              maxptr[maxctr]=ctr;
                              maxhrvtpatemp=hrvtpa[ctr];
                              saveobj=presentobj;
                    //  for (fixresctr=0; fixresctr<lines; fixresctr++)
                    //      saveht[fixresctr]=resht[fixresctr];
                    }
                    else if (sumdev==minsumdev[maxctr])
                    {
                        if (obj>maxobj[maxctr])
                        {
                            maxobj[maxctr]=obj;
                            maxptr[maxctr]=ctr;
                            maxhrvtpatemp=hrvtpa[ctr];
                            saveobj=presentobj;
                        //  for (fixresctr=0; fixresctr<lines; fixresctr++)
                        //  saveht[fixresctr]=resht[fixresctr];
                        }
                    }
                    if (per==6 && maxptr[1]==23)
                        fprintf(editfp,"");

                }

                zerosflag=1;

            } while (hrvtpa[ctr]<tpa[ctr]);

            hrvtpa[ctr]=0;
            hrvflag=0;
            lastctr=ctr;

        }   // end of if statement, no max ctr

        solflag=1;

    }

    if (((minsumdev[maxctr-1]>minsumdev[maxctr] || maxctr==0)
        || (minsumdev[maxctr-1]==0 && minsumdev[maxctr]==0 &&
maxobj[maxctr]>maxobj[maxctr-1]))
        && maxptr[maxctr]!=99999999)
    {
        maxhrvtpa[maxptr[maxctr]]=maxhrvtpatemp;
        hrvtpa[maxptr[maxctr]]=maxhrvtpatemp;
    }
    maxhrvtpatemp=0;

// stabilizing
   ***********************************************************************
**********
    for (solctr=0; solctr<lines; solctr++)
        stblhrvtpa[solctr]=maxhrvtpa[solctr];
```

```
                    if (per==18)
                        printf("");

                    if (((minsumdev[maxctr-1]>minsumdev[maxctr] || maxctr==0)
                        || (minsumdev[maxctr-1]==0 && minsumdev[maxctr]==0 &&
maxobj[maxctr]>maxobj[maxctr-1]))
                        && maxptr[maxctr]!=99999999)
                    {   do
                        {   for (stblctr=0; stblctr<maxctr+1; stblctr++)
                            {

                                stblhrvtpa[maxptr[stblctr]]=0;

                                do
                                {
                                    for (fixresctr=0; fixresctr<lines; fixresctr++)
                                        restpa[fixresctr]=tpa[fixresctr]-
stblhrvtpa[fixresctr];

                                    for (fixresctr=0; fixresctr<lines; fixresctr++)
                                    {   resba[fixresctr]=ba[ctr];
                                        if (restpa[fixresctr]==0)
                                            resba[fixresctr]=0;
                                    }


                                    if (stblflag==1)
                                    {   fprintf(editfp,"");
                                    }
                                    volharvest(stblhrvtpa,dbh,ht);
                                    vols(stblhrvtpa, dbh, dg, sp, ht);
                                    presentobj=objfunction(stblhrvtpa,dbh,vol);

  presentsumdev=sumdevfn(sp,ba,presentpba,cw,dbh,restpa,morttpa,
                                        0,presentobj,ht,cr,flameln,editfp);

                                    harvesttrees=0;
                                    totaltrees=0;
                                    treestimesdbh=0;
                                    hrvtreestimesdbh=0;
                                    for (fixresctr=0; fixresctr<lines; fixresctr++)
                                    {
                                        harvesttrees+=stblhrvtpa[fixresctr];
                                        totaltrees+=restpa[fixresctr];

  treestimesdbh+=restpa[fixresctr]*resdbh[fixresctr];

  hrvtreestimesdbh+=stblhrvtpa[fixresctr]*resdbh[fixresctr];

                                    }
```

```
//   oldflameln=flameln;

     if (harvesttrees>0)
     {
          preavgdbh=treestimesdbh/totaltrees;
          hrvavgdbh=hrvtreestimesdbh/harvesttrees;
          dtodratio=hrvavgdbh/preavgdbh;
     }
     else
          dtodratio=0;

     if (dtodratio<0.9 && harvesttrees>0)
          flameln-=1;
     if (harvesttrees==0)
          flameln+=0.33;
     if (flameln<0.33)
          flameln=0.33;
     else if (flameln>15)
          flameln=15;

     for (fixresctr=0; fixresctr<lines; fixresctr++)
     {    olddbh[fixresctr]=dbh[fixresctr];
          oldht[fixresctr]=ht[fixresctr];
          oldcr[fixresctr]=cr[fixresctr];
          oldabirth[fixresctr]=abirth[fixresctr];

          if (ht[fixresctr]>4.5)

cw[fixresctr]=cwco1[sp[fixresctr]]*pow(dbh[fixresctr],cwco2[sp[fixresctr]]);
          else

cw[fixresctr]=cwco3[sp[fixresctr]]*ht[fixresctr];

     }
     oldflameln=flameln;

     for (growctr=0; growctr<growperiods; growctr++)
     {
          growtrees(restpa,dbh,ht);

          for (fixresctr=0; fixresctr<lines; fixresctr++)
          {    dbh[fixresctr]=resdbh[fixresctr];
               ht[fixresctr]=resht[fixresctr];
               cr[fixresctr]=rescr[fixresctr];
          }
     }

     pba=0;
     for (fixresctr=0; fixresctr<lines; fixresctr++)
     {
```

```
                                        pba+=futba[fixresctr]*restpa[fixresctr];
                                        if (ht[fixresctr]>4.5)

cw[fixresctr]=cwco1[sp[fixresctr]]*pow(dbh[fixresctr],cwco2[sp[fixresctr]]);
                                        else

cw[fixresctr]=cwco3[sp[fixresctr]]*ht[fixresctr];
                                    }



                                    vols(restpa, resdbh, dg, sp, resht);

futureobj=objfunction(restpa,resdbh,vol)/(pow((1+interestrate),(per+1)*5));
                                    obj=presentobj+futureobj;


futuresumdev=sumdevfn(sp,futba,pba,rescw,resdbh,restpa,morttpa,
                                        1,futureobj,resht,rescr,flameln,editfp);
                                    sumdev=futuresumdev+presentsumdev;


                                    for (fixresctr=0; fixresctr<lines; fixresctr++)
                                    {
                                        dbh[fixresctr]=olddbh[fixresctr];
                                        ht[fixresctr]=oldht[fixresctr];
                                        cr[fixresctr]=oldcr[fixresctr];
                                        abirth[fixresctr]=oldabirth[fixresctr];

                                        if (ht[fixresctr]>4.5)

cw[fixresctr]=cwco1[sp[fixresctr]]*pow(dbh[fixresctr],cwco2[sp[fixresctr]]);
                                        else

cw[fixresctr]=cwco3[sp[fixresctr]]*ht[fixresctr];

                                    }
                                    flameln=oldflameln;


                                    if (obj>minobjvalue)
                                    {

                                        if (gcctr>=5)
                                        // if (gcctr<7)  // use PNV for all runs
                                        {
                                            if ((sumdev-k*obj)<minsumdev[stblctr])
                                            {
                                                minsumdev[stblctr]=sumdev-k*obj;
                                                maxobj[stblctr]=obj;
                                        savesumdev=sumdev;

    maxhrvtpatemp=stblhrvtpa[maxptr[stblctr]];
                                        minflag=1;
                                        ctrformaxhrvtemp=maxptr[stblctr];
                                        saveobj=presentobj;
                                //      for (fixresctr=0; fixresctr<lines;
fixresctr++)
                                //      saveht[fixresctr]=resht[fixresctr];
                                    }
                                //  else if ((sumdev-
k*obj)>=(minsumdev[stblctr]-0.001) && // no longer works for large numbers
                                //      (sumdev-
k*obj)<=(minsumdev[stblctr]+0.001))     // use .0001% instead
                                    else if (fabs(sumdev-
k*obj)>=fabs(minsumdev[stblctr]-0.0001*minsumdev[stblctr]) &&
                                        fabs(sumdev-
k*obj)<=fabs(minsumdev[stblctr]+0.0001*minsumdev[stblctr]))
                                    {
                                        if (obj>=(maxobj[stblctr]-0.01))
                                        {
                                            maxobj[stblctr]=obj;
                                            savesumdev=sumdev;

    maxhrvtpatemp=stblhrvtpa[maxptr[stblctr]];
                                            minflag=1;
                                            ctrformaxhrvtemp=maxptr[stblctr];
                                            saveobj=presentobj;
                                    //      for (fixresctr=0; fixresctr<lines;
fixresctr++)
                                    //          saveht[fixresctr]=resht[fixresctr];
                                        }
                                    }
                                }
                                else if (sumdev<minsumdev[stblctr])
                                {   minsumdev[stblctr]=sumdev;
                                    maxobj[stblctr]=obj;
                                    savesumdev=sumdev;
                                    maxhrvtpatemp=stblhrvtpa[maxptr[stblctr]];
                                    minflag=1;
                                    ctrformaxhrvtemp=maxptr[stblctr];
                                    saveobj=presentobj;
                                //  for (fixresctr=0; fixresctr<lines; fixresctr++)
                                //      saveht[fixresctr]=resht[fixresctr];
                                }
                                else if (sumdev>=(minsumdev[stblctr]-0.001) &&
                                    sumdev<=(minsumdev[stblctr]+0.001))
                                {   if (obj>=(maxobj[stblctr]-0.01))
                                    {   maxobj[stblctr]=obj;
                                        savesumdev=sumdev;
```

```
     maxhrvtpatemp=stblhrvtpa[maxptr[stblctr]];
                                                  minflag=1;
                                                  ctrformaxhrvtemp=maxptr[stblctr];
                                                  saveobj=presentobj;
                                            //    for (fixresctr=0; fixresctr<lines;
fixresctr++)
                                            //          saveht[fixresctr]=resht[fixresctr];
                                               }
                                            }
                                         }

                                         stblhrvtpa[maxptr[stblctr]]+=HARVESTNUMBER;
     //David defined in var.h//5;

                                         if (stblhrvtpa[maxptr[stblctr]]>tpa[maxptr[stblctr]]
&& stblflag==0)
                                         {   stblhrvtpa[maxptr[stblctr]]=tpa[maxptr[stblctr]];
                                             stblflag=1;
                                         }


                                      }while
(stblhrvtpa[maxptr[stblctr]]<=tpa[maxptr[stblctr]]);//-1)

                                      stblflag=0;

                                      stblhrvtpa[maxptr[stblctr]]=maxhrvtpatemp;

                                   }

                                   maxhrvtpatemp=0;
                                   for (fixresctr=0; fixresctr<lines; fixresctr++)
                                      maxhrvtpa[fixresctr]=stblhrvtpa[fixresctr];

                                   fprintf(growfp,"stabilizing: stblctr=%d mindev=%f",stblctr-
1,minsumdev[stblctr-1]);
                             //    minflag=0;
                          //    }
// standard deviation to see if the harvest tpa have stabilized
********************************
                                   sumobj=0;
                                   sumsqterm=0;

                                   for(sdctr=0; sdctr<maxctr+1; sdctr++)
                                      sumobj+=minsumdev[sdctr];

                                   meanobj=sumobj/(maxctr+1);

                                   for(sdctr=0; sdctr<maxctr+1; sdctr++)
```

```
                        {    sqterm=pow((minsumdev[sdctr]-meanobj),2);
                             sumsqterm+=sqterm;
                        }

                        stdev=pow(sumsqterm/(maxctr),0.5);
                        fprintf(growfp," stdev=%f\n",stdev);
                        if (stdev>0.5)
                             stdevctr++;

                   //   if (maxctr==0)
                   //        stdev=0;

                        if (stdevctr>6)
                             printf("might want to check\n");

                        if (stdev<0)
                             stdev=0;

                   } while (stdev>0.5);
                   stdevctr=0;
              }

              for (stblctr=0; stblctr<lines; stblctr++)
              {
                   maxhrvtpa[stblctr]=stblhrvtpa[stblctr];
                   hrvtpa[stblctr]=stblhrvtpa[stblctr];
              }

              maxctr+=1;
              if (maxctr>=24)
                   printf("found");

              fprintf(growfp,"\n");
         }   while (minsumdev[maxctr-1]<lastdev
                   || (minsumdev[maxctr-1]==0 && lastdev==0 && maxobj[maxctr-
1]>lastobj));


   // grow with best prescription

         if (mcflimit==1)
         {
              totalmcfharvest=0;
              volharvest(maxhrvtpa,dbh,ht);
              for (fixresctr=0; fixresctr<lines; fixresctr++)
                   totalmcfharvest+=vol[fixresctr]*maxhrvtpa[fixresctr];

              if (totalmcfharvest/1000<0.5)     // change mcflimit
              {
```

```
            for (fixresctr=0; fixresctr<lines; fixresctr++)
                maxhrvtpa[fixresctr]=0;
        }

    }


    for (fixresctr=0; fixresctr<lines; fixresctr++)
        restpa[fixresctr]=tpa[fixresctr]-maxhrvtpa[fixresctr];

    growtrees(restpa,dbh,ht);

    volharvest(maxhrvtpa,dbh,ht);
    presentobj=objfunction(maxhrvtpa,dbh,vol);
//  fprintf(hsfp,"%4.2f %4.2f ",mcfvolharvest,baharvest);

    if (per==15)
        printf("");
    vegclass=vegclassification(dbh,tpa,cw,reportsp,ba);
    presentsumdev=sumdevfn(sp,ba,presentpba,cw,dbh,tpa,
        morttpa,2,presentobj,ht,cr,flameln,editfp);
    if (GoalEmp==9)
        presentsumdev=sumdevfn(sp,resba,presentpba,rescw,resdbh,restpa,
            morttpa,2,presentobj,resht,rescr,flameln,editfp);

    // this next part on changing flameln needs to be checked with Bernie <>
    // it also occurs in hrvtpa, stblhrvtpa

    harvesttrees=0;
    totaltrees=0;
    treestimesdbh=0;
    hrvtreestimesdbh=0;
    mcfvolharvest=0;
    baharvest=0;
    for (fixresctr=0; fixresctr<lines; fixresctr++)
    {
        harvesttrees+=maxhrvtpa[fixresctr];
        if (maxhrvtpa[fixresctr]>0)
            mcfvolharvest+=maxhrvtpa[fixresctr]*vol[fixresctr];
        baharvest+=maxhrvtpa[fixresctr]*ba[fixresctr];
        totaltrees+=tpa[fixresctr];
        treestimesdbh+=tpa[fixresctr]*dbh[fixresctr];
        hrvtreestimesdbh+=maxhrvtpa[fixresctr]*dbh[fixresctr];

    }

    fprintf(hsfp,"%4.2f %4.2f ",mcfvolharvest,baharvest);

    if (harvesttrees>0)
    {
```

```
                    preavgdbh=treestimesdbh/totaltrees;
                    hrvavgdbh=hrvtreestimesdbh/harvesttrees;
                    dtodratio=hrvavgdbh/preavgdbh;
                }
                else
                    dtodratio=0;

                if (dtodratio<0.9 && harvesttrees>0)
                    flameln-=1;
                if (harvesttrees==0)
                    flameln+=0.33;
                if (flameln<0.33)
                    flameln=0.33;
                else if (flameln>15)
                    flameln=15;


    // print harvest
trees*************************************************************************
***

                for (ctr=0; ctr<lines; ctr++)
                {
                    if (maxhrvtpa[ctr]>0)
                        fprintf(growfp,"tpa harvested %8.2lf from
%2d\n",maxhrvtpa[ctr],ctr);
                }

    // new cycle
*****************************************************************************
********
                plotba=0;
                snagscost=0;
                goalattainment[per]=minsumdev[0];
                presentnetvalue[per]=maxobj[0];
                totalsumdev+=savesumdev;
                savesumdev=0;
                totalvalue+=saveobj/(pow((1+interestrate),(per*5)));
                fprintf(hsfp,"%4.2f ",saveobj/(pow((1+interestrate),(per*5))));
                saveobj=0;
                flagforperiod=1; // this allows sort in growtrees once per period
                flagforhtlc=1;

    // if regen, enter new trees here and reset lines

                for (ctr=0; ctr<lines; ctr++)
                {
                    if (per==0)
                        morttpa[ctr]=origmorttpa[ctr];
                    else
                morttpa[ctr]=0;
            tpa[ctr]=tpa[ctr]-maxhrvtpa[ctr];
            if (tpa[ctr]<0)
                tpa[ctr]=tpa[ctr];
            ba[ctr]=pow((resdbh[ctr]/24),2)*pi;
            baptr[ctr]=ctr;
            if (ld[ctr]==0 && per==0)
            {
                ht[ctr]=ht[ctr];
                dbh[ctr]=dbh[ctr];
            }
            else
            {
                dbh[ctr]=resdbh[ctr];
                ht[ctr]=resht[ctr];
            }
            htorder[ctr]=ctr;
            cr[ctr]=rescr[ctr];

            maxobj[ctr]=0;
            maxhrvtpa[ctr]=0;
            stblhrvtpa[ctr]=0;
            maxptr[ctr]=99999999;
            hrvtpa[ctr]=0;
            minsumdev[ctr]=1.7*pow(10,308);
            plotba+=ba[ctr]*tpa[ctr];
            if (dbh[ctr]==0)
                dbh[ctr]=0;

            if (ht[ctr]>4.5)
                cw[ctr]=cwco1[sp[ctr]]*pow(dbh[ctr],cwco2[sp[ctr]]);
            else
                cw[ctr]=cwco3[sp[ctr]]*ht[ctr];

        }
        lastdev=1.7*pow(10,308);

        vegclass=vegclassification(dbh,tpa,cw,reportsp,ba);
        fprintf(hsfp," %d ",vegclass);

        if (createsnags)
        {
            for(ctr=0; ctr<lines; ctr++)
            {
                morttpa[ctr]+=createsnagsmorttpa[ctr];        // change added a
plus
                createsnagsmorttpa[ctr]=0;
            }
        }
```

```
// this section for mortality

            if (elev>4000 && plotba>120 && aspect>=45 && aspect<=315)
            {                                               // if this is possible for
windthrow

                quicksort3(htorder,0,lines-1,ht);

                ctr=0;
                talltrees=0;
                while (talltrees<5 && ctr<lines)
                {
                    talltrees+=tpa[htorder[ctr]];
                    heighttall5trees+=ht[htorder[ctr]]*tpa[htorder[ctr]];
                    if (talltrees>5)
                        heighttall5trees-=ht[htorder[ctr]]*(talltrees-5);
                    ctr++;
                }
                if (talltrees>0)
                    heighttall5trees/=talltrees;
                else
                    heighttall5trees=0;

                if (heighttall5trees>50)
                {
                    mortba=0;
                    windmortba=plotba*0.0025;
                    ctr=0;

                    do
                    {   mortba+=ba[htorder[ctr]]*tpa[htorder[ctr]];
                        if (mortba<windmortba)
                            windmorttpa=tpa[htorder[ctr]];
                        else
                        {   windmorttpa=tpa[htorder[ctr]]-(mortba-
windmortba)/ba[htorder[ctr]];
                        //  mortba-=mortba-windmortba;
                            mortba=windmortba;
                        }
                        morttpa[htorder[ctr]]+=windmorttpa;
                        tpa[htorder[ctr]]-=windmorttpa;
                        ctr++;
                        if (ctr==lines)
                            mortba=windmortba;
                    } while (mortba<windmortba);
                }
            }

            quicksort(baptr,0,lines-1,ba);
```

```
if (plotba>0)
{
    morttot=plotba*0.0005;
    lpinemort=morttot*0.67;
    mpinemort=morttot-lpinemort;

    lpinekill=0;
    mpinekill=0;
    lpinetpa=0;
    mpinetpa=0;
    ctr=0;

    while (dbh[baptr[ctr]]>8 && ctr<lines)
    {
        if (sp[baptr[ctr]]==3 || sp[baptr[ctr]]==5 || sp[baptr[ctr]]==7)
        {
            if (dbh[baptr[ctr]]>20)
            {
                if (lpinekill < lpinemort)
                {
                    lpinekill+=tpa[baptr[ctr]]*ba[baptr[ctr]];
                    lpinetpa+=tpa[baptr[ctr]];
                    morttpa[baptr[ctr]]+=tpa[baptr[ctr]];
                    tpa[baptr[ctr]]-=tpa[baptr[ctr]];

                    if (lpinekill>lpinemort)
                    {   lpinetpa=(lpinekill-lpinemort)/ba[baptr[ctr]];
                        morttpa[baptr[ctr]]-=lpinetpa;
                        tpa[baptr[ctr]]+=lpinetpa;
                    }
                }
            }
            else if (mpinekill < mpinemort)
            {
                mpinekill+=tpa[baptr[ctr]]*ba[baptr[ctr]];
                mpinetpa+=tpa[baptr[ctr]];
                morttpa[baptr[ctr]]+=tpa[baptr[ctr]];
                tpa[baptr[ctr]]-=tpa[baptr[ctr]];

                if (mpinekill>mpinemort)
                {   mpinetpa=(mpinekill-mpinemort)/ba[baptr[ctr]];
                    morttpa[baptr[ctr]]-=mpinetpa;
                    tpa[baptr[ctr]]+=mpinetpa;
                }
            }
        }

        ctr++;
    }
```

```
        if ((pag==5 || pag==6 || pag==7)&& rootdiseasetrigger==1)
        rootdisease(vol,sp);

        for (ctr=0; ctr<lines; ctr++)
        {
            if (morttpa[ctr]>0)
                fprintf(growfp,"morttpa[%d]=%4.2f\n",ctr,morttpa[ctr]);
        }
    }

    deadwood(morttpa,ht,dbh);

    if (plotba>0)
    {
        snags=0;
        snagtrees=0;
        for (a=0; a<lines; a++)
        {
            if (dbh[a]>=16 && dbh[a]<30)
                snagtrees+=tpa[a];
        }

        for (a=2; a<5; a++)
            for (b=0; b<4; b++)
                snags+=reportsnag[a][b];

        fprintf(growfp,"snags=%4.2f\n",snags);

        if (snags<snagsgoal)
        {
            snagsgoaltemp=snagsgoal+0.3;
            if (createsnags && (snags+snagtrees>snagsgoaltemp))
            {
                nusnags=snagsgoaltemp-snags;
                snagscost=34.75*nusnags;
                reportnusnags=0;


                ctr=0;
                while (snags<snagsgoaltemp)
                {
                    if (dbh[ctr]>=16 && dbh[ctr]<30)
                    {
                        snags+=tpa[ctr];
                        if (snags<snagsgoaltemp)
                            snagsmorttemp=tpa[ctr];
                        else
                        {
                            snagsmorttemp=tpa[ctr]-(snags-snagsgoaltemp);
```

```
                                        snags=snagsgoaltemp;
                                    }
                                    tpa[ctr]-=snagsmorttemp;
                                    createsnagsmorttpa[ctr]=snagsmorttemp;
                                    reportnusnags+=snagsmorttemp;
                                    fprintf(growfp,"snags created
[%d]=%6.2f\n",ctr,snagsmorttemp);
                                }
                                ctr++;
                                if (ctr==lines)
                                    snags=snagsgoaltemp;
                            }

                            fprintf(hsfp," %4.2f",reportnusnags);
                        }
                    }
                }

//      deadwood(morttpa,ht,dbh);
//      presentsumdev=sumdevfn(sp,ba,presentpba,cw,dbh,tpa,
//          morttpa,2,presentobj,ht,cr,flameln,editfp);

        oldlines=lines;
        if (per>1 && per%2==0)
            regen(rsavetpa,rsaveba,rsavesp,rsavemaxhrvtpa,rsavelines);
        if (lines>oldlines)
        {
            for (ctr=0; ctr<lines; ctr++)
                baptr[ctr]=ctr;
        }
        if (per%2==0)//((totaltrees-harvesttrees)/totaltrees > 0.20 &&
regenper!=per+1)
        {
            regenper=per+2;
            rsavelines=lines;
            for (fixresctr=0; fixresctr<lines; fixresctr++)
            {
                rsavetpa[fixresctr]=tpa[fixresctr];
                rsavesp[fixresctr]=sp[fixresctr];
                rsaveba[fixresctr]=ba[fixresctr];
                rsavemaxhrvtpa[fixresctr]=maxhrvtpa[fixresctr];
            }
        }


        fprintf(growfp,"total deviations %7.2f, total value
%9.2f\n",totalsumdev,totalvalue);

        fprintf(growfp,"Plot Species TPA   DBH    HT  CR  CW   beg of per
%d\n",per+1);
```

```
                    for (ctr=0; ctr<lines; ctr++)
                        fprintf(growfp,"%d%d%6d%9.2f%7.2f%7.1f%4.0f %8.2f
%d\n",per+1,plot[ctr],sp[ctr],tpa[ctr],
                        dbh[ctr],ht[ctr],cr[ctr],cw[ctr],ctr);
                    fprintf(growfp,"\n");
                    fprintf(hsfp,"\n");

                    fprintf(davidfp,"Period\n%d %d %d\n",per+1,lines,vegclass);
                    for (ctr=0; ctr<lines; ctr++)
                        fprintf(davidfp,"%d %f %.2f %.1f %.0f %.2f\n",sp[ctr],tpa[ctr],
                            dbh[ctr],ht[ctr],cr[ctr],cw[ctr]);

                } // end of periods

                fprintf(growfp,"total deviations %7.2f total value
%9.2f",totalsumdev,totalvalue);
                fprintf(growfp," %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f",tinsectdev,
                    tverticaldev, tsnagsdev, tdwddev, tfbi, tfei, tfishtreesdev);

            } // end of goal combos


        } /* end of section for Path algorithm*/

// if the dll is loaded, unload it
    if( DLL )
    {
        FreeLibrary( DLL );
    }

} //end of void main(void)
*********************************************************************

BFVOL
#include "var.h"
int numlog;
double frmcls[totsp], smdia[20], glogln[20], bfrevenue;
double bfmind[totsp]={9,9,9,9,9,9,9,9,9,9};
double bftopd[totsp]={6,6,6,6,6,6,6,6,6,6};

double bfvol(int sp, double dbh, double ht, double brat)
{
    int bfctr, diam;
    double bbfv, td, dib, fc, tlog, factor;

    double logprice[51][10]=
    {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0, 269, 500, 345,   0, 345, 329, 363,   0, 329,
        0, 269, 477, 345,   0, 345, 329, 363,   0, 329,
        0, 269, 454, 345,   0, 345, 329, 363,   0, 329,
        0, 269, 431, 345,   0, 345, 329, 363,   0, 329,
        0, 269, 408, 345,   0, 345, 329, 363,   0, 329,
        0, 269, 385, 345,   0, 345, 329, 363,   0, 329,
        0, 269, 379, 345,   0, 345, 329, 363,   0, 329,
        0, 284, 373, 345,   0, 345, 331, 361,   0, 331,
        0, 302, 367, 344,   0, 344, 333, 360,   0, 333,
        0, 317, 361, 344,   0, 344, 330, 360,   0, 330,
        0, 331, 358, 345,   0, 345, 328, 360,   0, 328,
        0, 343, 357, 348,   0, 348, 318, 363,   0, 318,
        0, 356, 357, 352,   0, 352, 307, 366,   0, 307,
        0, 366, 357, 355,   0, 355, 297, 368,   0, 297,
        0, 375, 361, 357,   0, 357, 287, 370,   0, 287,
        0, 381, 365, 360,   0, 360, 281, 373,   0, 281,
        0, 388, 369, 363,   0, 363, 275, 375,   0, 275,
        0, 392, 373, 366,   0, 366, 273, 378,   0, 273,
        0, 396, 378, 370,   0, 370, 271, 382,   0, 271,
        0, 396, 383, 374,   0, 374, 261, 385,   0, 261,
        0, 396, 388, 378,   0, 378, 252, 389,   0, 252,
        0, 393, 394, 382,   0, 382, 255, 393,   0, 255,
        0, 391, 399, 386,   0, 386, 257, 397,   0, 257,
        0, 385, 404, 390,   0, 390, 252, 400,   0, 252,
        0, 378, 409, 394,   0, 394, 248, 404,   0, 248,
        0, 368, 414, 398,   0, 398, 247, 409,   0, 247,
        0, 357, 419, 402,   0, 402, 246, 413,   0, 246,
        0, 343, 422, 406,   0, 406, 246, 417,   0, 246,
        0, 330, 422, 410,   0, 410, 246, 421,   0, 246,
        0, 312, 422, 415,   0, 415, 246, 426,   0, 246,
        0, 293, 422, 421,   0, 421, 246, 432,   0, 246,
        0, 293, 422, 428,   0, 428, 245, 439,   0, 245,
        0, 293, 422, 435,   0, 435, 245, 447,   0, 245,
        0, 293, 422, 444,   0, 444, 245, 456,   0, 245,
        0, 293, 422, 453,   0, 453, 245, 465,   0, 245,
        0, 293, 422, 463,   0, 463, 245, 475,   0, 245,
        0, 293, 422, 472,   0, 472, 245, 485,   0, 245,
        0, 293, 422, 482,   0, 482, 245, 495,   0, 245,
        0, 293, 422, 492,   0, 492, 245, 405,   0, 245,
        0, 293, 422, 501,   0, 501, 245, 515,   0, 245,
        0, 293, 422, 510,   0, 510, 245, 525,   0, 245,
        0, 293, 422, 520,   0, 520, 245, 535,   0, 245,
        0, 293, 422, 530,   0, 530, 245, 545,   0, 245,
        0, 293, 422, 540,   0, 540, 245, 555,   0, 245,
        0, 293, 422, 550,   0, 550, 245, 565,   0, 245};

bbfv=0;
bfrevenue=0;
```

```
    if (dbh>bfmind[sp])
    {
        td=bftopd[sp]*brat;
        dib=dbh*brat;
        fc=formcl(sp,dbh);
        rxdibs(dbh,fc,ht,td);

        tlog=0;
        for (bfctr=0; bfctr<=numlog; bfctr++)
        {
            factor=scalef(smdia[bfctr],glogln[bfctr]);
            tlog+=glogln[bfctr];
            bbfv+=glogln[bfctr]*factor;
            diam=int(smdia[bfctr]+0.5);
            if (diam>50)
                diam=50;
            bfrevenue+=(logprice[diam][sp]/1000*glogln[bfctr]*factor);
        }
    }

    return (bbfv);

}

BRATIO
#include "var.h"

double bratio(int sp, double dbh)
{   double brat, bark, dib;

    double bratco1[totsp]={-0.3386,0.1045,-
0.0549,0.4448,0.1429,0.4448,0.1593,1.9064,2.0754,0.1593};
    double
bratco2[totsp]={0.0438,0.1161,0.1626,0.1033,0.1137,0.1033,0.1089,0.0172,0.0204,0.1089
};
    double bratco3[totsp]={2.5626,0,0,0,0,0,0,-8.4312,-14.0088,0};

    // equation not good for trees <3.9 in DBH
    if (dbh<3.9)
        dbh=3.9;
    bark=bratco1[sp] +bratco2[sp]*dbh +bratco3[sp]/dbh;
    if (bark<0.1)
        bark=0.1;

    dib=dbh-bark;
    brat=dib/dbh;
    if (brat >= 1 || brat <=0)
        brat=0.99;
    return (brat);

}
DEADWOOD
#include "var.h"

#include "windows.h"
#define TAPER_DLL
#include "taper.h"
extern diDIB        pfdiDIB;

extern int lines, per, deadlines;
extern int snagsp[linesor][periods],dwdsp[linesor][periods][periods];
extern int snagflag[linesor][periods], dwdage[linesor][periods][periods],
snagage[linesor][periods];
extern int dwdflag[linesor][periods][periods][2], sp[linesor];
extern double snagtpa[linesor][periods], snaght[linesor][periods];
extern double snagdensity[linesor][periods], snagdbh[linesor][periods], sdensity[linesor];
extern double dwdln[linesor][periods][periods][2], dwdld[linesor][periods][periods][2];
extern double dwdpa[linesor][periods][periods][2], dwddensity[linesor][periods][periods];
extern double reportsnag[5][5], reportdwd[2][5][5];
extern double cr[linesor];

extern FILE *hsfp;
void deadwood(double morttpa[], double ht[], double dbh[])
{
    int dctr, dper, actr, lctr, diam, cls, ln;
    int tooshort=0, toosmall=0, toodead=0;
    int snagdclass, state, snagagetemp, sptemp;
    int dwdclass, dwdstate;
    double snaghtl, snaghttemp, snagfall, snagfalltemp, snagdensitytemp;
    double snagupperd, snagstumpd, snagmaxagetemp, dwdmaxagetemp;

    int dwdmaxage[3]={320,195,75};
    double snagdensityrate[3]={0.017,0.033,0.053};
    double snagfallco[3][3]={0.008, 0.008, 0.008,
                             0.013, 0.022, 0.027,
                             0.021, 0.043, 0.054};
    double snaghtlossco[3][3]={0.019, 0.019, 0.019,
                             0.032, 0.038, 0.042,
                             0.0421,0.056, 0.063};
    double snagfalllag[3]={15,10,0};
    double snagdecaylag[3]={0.9,1.4,1.25};
    double logdecayrate[2][3]={0.012, 0.015, 0.026,
                        0.008, 0.01, 0.02};
    double snagmindensity[3]={0.007,0.013,0.021};
    double snagmaxage[3]={260,135,80};
    double dwdldred[3]={0.0031,0.0037,0.004};
    double dwdlnred[3]={0.0026,0.003,0.004};
    double dwdiv[3]={70,60,35};
    double sphtl[totsp],spsnagfall[totsp],spsnagdecay[totsp],spsnagmaxage[totsp];
    double spldred[totsp],splnred[totsp],spdwddecay[totsp],spdwdmaxage[totsp];
    //FILE *snagfp, *dwdfp;

    //snagfp=fopen("c:\\apple\\vegtype\\snag.txt","w");
    //snagfp=fopen("c:\\apple\\vegtype\\dwd.txt","w");

//  pfdiDIB         = (diDIB)GetProcAddress(DLL,"diDIB");



    for(dctr=0; dctr<totsp; dctr++)
    {
        sphtl[dctr]=1;
        spsnagfall[dctr]=1;
        spsnagdecay[dctr]=1;
        spsnagmaxage[dctr]=1;
        spldred[dctr]=1;
        splnred[dctr]=1;
        spdwddecay[dctr]=1;
        spdwdmaxage[dctr]=1;
    }

    for(dctr=0; dctr<lines; dctr++)
    {
        if (morttpa[dctr]>0 && ht[dctr]>=30)
        {
            snagtpa[dctr][per]=morttpa[dctr];
            snagflag[dctr][per]=1;
            snaght[dctr][per]=ht[dctr];
            if (sdensity[dctr]>0 && per==0)
                snagdensity[dctr][per]=sdensity[dctr];
            else
                snagdensity[dctr][per]=0.452;
            if (per==0)
                snagage[dctr][per]=snagage[dctr][0];
            else
                snagage[dctr][per]=0;
            snagdbh[dctr][per]=dbh[dctr];
            snagsp[dctr][per]=sp[dctr];
        }
    }

    for(dper=0; dper<5; dper++)
        for (actr=0; actr<5; actr++)
        {
            reportsnag[dper][actr]=0;
            for (lctr=0; lctr<2; lctr++)
                reportdwd[lctr][dper][actr]=0;
        }
    for(dctr=0; dctr<deadlines; dctr++)
    {
        for(dper=0; dper<per+1; dper++)
        {
            if (snagflag[dctr][dper]==1)
            {
                if (snagdbh[dctr][dper]<15)
                    snagdclass=2;
                else if (snagdbh[dctr][dper]<25)
                    snagdclass=1;
                else
                    snagdclass=0;

                if (snagdensity[dctr][dper]>=0.284)
                    state=0;
                else if (snagdensity[dctr][dper]>=0.197)
                    state=1;
                else
                    state=2;

                if (snagage[dctr][dper]>snagfalllag[snagdclass])
                {                       // htl is actually ht remaining, temp is part that was lost
                    snaghtl=snaght[dctr][dper]*pow((1-
snaghtlossco[snagdclass][state]),5);
                    snaghtl*=sphtl[snagsp[dctr][dper]];
                    snaghttemp=snaght[dctr][dper]-snaghtl;
                    snaght[dctr][dper]=snaghtl;
                                        // snagfall is actually amount standing, temp is # that fell
                    snagfall=snagtpa[dctr][dper]*exp(snagfallco[snagdclass][state]*-5);
                    snagfall*=spsnagfall[snagsp[dctr][dper]];
                    snagfalltemp=snagtpa[dctr][dper]-snagfall;
                    snagtpa[dctr][dper]=snagfall;

                    if(snagdbh[dctr][dper]>0)
                    {
                        snagupperd = //snagdbh[dctr][dper]*0.3;
                            pfdiDIB( WALTERS_HANN,  // walters hann (organon)
taper functions
                                        0,                  // species, see taper.h
                                            snagdbh[dctr][dper],            // dbh
                                            ht[dctr],       // total height
                                            cr[dctr],       // crown ratio
                                            snaghtl );          // h


    snagupperd=snagupperd/bratio(snagsp[dctr][dper],snagdbh[dctr][dper]);

                        snagstumpd = //snagdbh[dctr][dper];
```

```
                                        pfdiDIB( WALTERS_HANN,
                                        0,
                                        snagdbh[dctr][dper],
                                        ht[dctr],
                                        cr[dctr],
                                        1.0 );


snagstumpd=snagstumpd/bratio(snagsp[dctr][dper],snagdbh[dctr][dper]);
                    //   snagstumpd=snagdbh[dctr][dper];
                    }

                    snagdensitytemp=snagdensity[dctr][dper];
                }


                if (snagage[dctr][dper]>snagdecaylag[snagdclass])
                {
                    snagdensity[dctr][dper]=snagdensity[dctr][dper]*
                        exp(snagdensityrate[snagdclass]*-5);
                    snagdensity[dctr][dper]*=spsnagdecay[snagsp[dctr][dper]];
                }

                snagagetemp=snagage[dctr][dper];
                snagmaxagetemp=snagmaxage[snagdclass];
                snagmaxagetemp*=spsnagmaxage[snagsp[dctr][dper]];
                sptemp=snagsp[dctr][dper];
            //  snagage[dctr][dper]+=5;
                if (snagdensity[dctr][dper]<snagmindensity[snagdclass] ||
                    snagage[dctr][dper]>snagmaxagetemp)
                    snagflag[dctr][dper]=0;

            }

            if (snagdensitytemp>0.14)
            {
                dwdage[dctr][per][dper]=snagagetemp;
                dwddensity[dctr][per][dper]=snagdensitytemp;
                dwdsp[dctr][per][dper]=sptemp;

                if (snaghttemp>0)
                {
                    dwdln[dctr][per][dper][1]=snaghttemp;
                    dwdpa[dctr][per][dper][1]=snagtpa[dctr][dper];
                    dwdld[dctr][per][dper][1]=snagupperd;
                    dwdflag[dctr][per][dper][0]=1;
                }

                if (snagfalltemp>0)
                {
                    dwdln[dctr][per][dper][0]=snaght[dctr][dper]+snaghttemp-1; // does it
break off about 1foot?
                    dwdpa[dctr][per][dper][0]=snagfalltemp;
                    dwdld[dctr][per][dper][0]=snagstumpd;
                    dwdflag[dctr][per][dper][0]=1;
                }

            //  fprintf(hsfp,"per %d tops %4.2f logs %4.2f diam %4.2f
",per,snaghttemp*snagtpa[dctr][dper],
            //      (snaght[dctr][dper]+snaghttemp-
1)*snagfalltemp,dwdld[dctr][per][dper][0]);
            //  fprintf(hsfp,"dctr %d per %d dper %d\n",dctr,per,dper);
            }
// redo dwdclass and state?
            for (actr=0; actr<per+1; actr++)
            {
                if (dwddensity[dctr][actr][dper]>=0.284)
                    dwdstate=0;
                else
                    dwdstate=1;

                for (lctr=0; lctr<2; lctr++)
                {
                    if (dwdflag[dctr][actr][dper][lctr]==1)
                    {
                        if (dwdld[dctr][actr][dper][lctr]>=15)
                            dwdclass=0;
                        else if (dwdld[dctr][actr][dper][lctr]>=6)
                            dwdclass=1;
                        else
                            dwdclass=2;

                        dwdmaxagetemp=dwdmaxage[dwdclass];
                        dwdmaxagetemp*=spdwdmaxage[dwdsp[dctr][actr][dper]];

                        if (dwdage[dctr][actr][dper]<dwdmaxagetemp)
                        {

                            dwdln[dctr][actr][dper][lctr]=dwdln[dctr][actr][dper][lctr]*
                                exp(dwdlnred[dwdclass]*(dwdage[dctr][actr][dper]-
dwdiv[dwdclass]));

    dwdln[dctr][actr][dper][lctr]*=splnred[dwdsp[dctr][actr][dper]];
                            dwdld[dctr][actr][dper][lctr]=dwdld[dctr][actr][dper][lctr]*
                                exp(dwdldred[dwdclass]*(dwdage[dctr][actr][dper]-
dwdiv[dwdclass]));

    dwdld[dctr][actr][dper][lctr]*=spldred[dwdsp[dctr][actr][dper]];
                        }
                        else
                        dwdflag[dctr][actr][dper][lctr]=0;
                    }
                }

                //dwdage[dctr][dper][actr]+=5;
                if (dwdflag[dctr][actr][dper][0]==1 || dwdflag[dctr][actr][dper][1]==1)
                {
                    dwddensity[dctr][actr][dper]=dwddensity[dctr][actr][dper]*
                        exp(logdecayrate[dwdstate][dwdclass]*-5);
                    dwddensity[dctr][actr][dper]*=spdwddecay[dwdsp[dctr][actr][dper]];
                }
            }

            snaghtl=0;
            snaghttemp=0;
            snagfall=0;
            snagfalltemp=0;
            snagdensitytemp=0;
            snagupperd=0;
            snagstumpd=0;
            snagagetemp=0;

        }
    }


    for(dctr=0; dctr<deadlines; dctr++)
    {
        for(dper=0; dper<=per+1; dper++)
        {
            if (snagflag[dctr][dper]==1)
            {
                snagage[dctr][dper]+=5;

                if (snagdbh[dctr][dper]>=25)
                    diam=4;
                else if (snagdbh[dctr][dper]>=20)
                    diam=3;
                else if (snagdbh[dctr][dper]>=16)
                    diam=2;
                else if (snagdbh[dctr][dper]>=11)
                    diam=1;
                else if (snagdbh[dctr][dper]>=7)
                    diam=0;
                else toosmall=1;

                if (snagdensity[dctr][dper]>=0.38)
                    cls=0;
                else if (snagdensity[dctr][dper]>=0.284)
                    cls=1;
                    else if (snagdensity[dctr][dper]>=0.197)
                        cls=2;
                    else if (snagdensity[dctr][dper]>=0.14)
                        cls=3;
                    else if (snagdensity[dctr][dper]>0.021 && snagdbh[dctr][dper]<15)
                        cls=4;
                    else if (snagdensity[dctr][dper]>0.13 && snagdbh[dctr][dper]>=15 &&
                        snagdbh[dctr][dper]<25)
                        cls=4;
                    else if (snagdensity[dctr][dper]>0.021 && snagdbh[dctr][dper]>=25)
                        cls=4;
                    else
                        toodead=1;

                    if (toosmall==0 && toodead==0)
                        reportsnag[diam][cls]+=snagtpa[dctr][dper];

    //              fprintf(editfp"%4.2f %4.2f
%4.2f\n",snagdbh[dctr][dper],snagtpa[dctr][dper],
    //                  snagdensity[dctr][dper]);

                    toosmall=0;
                    toodead=0;
                }

                for (actr=0; actr<per+1; actr++)
                {
                    if (dwdflag[dctr][dper][actr][0]==1 || dwdflag[dctr][dper][actr][1]==1)
                            dwdage[dctr][dper][actr]+=5;

                    for (lctr=0; lctr<2; lctr++)
                    {
                        if (dwdflag[dctr][dper][actr][lctr]==1)
                        {
                            if (dwdld[dctr][dper][actr][lctr]>=25)
                                diam=4;
                            else if (dwdld[dctr][dper][actr][lctr]>=20)
                                diam=3;
                            else if (dwdld[dctr][dper][actr][lctr]>=16)
                                diam=2;
                            else if (dwdld[dctr][dper][actr][lctr]>=11)
                                diam=1;
                            else if (dwdld[dctr][dper][actr][lctr]>=7)
                                diam=0;
                            else
                                toosmall=1;

                            if (dwddensity[dctr][dper][actr]>=0.38)
                                cls=0;
                            else if (dwddensity[dctr][dper][actr]>=0.284)
                        cls=1;
                    else if (dwddensity[dctr][dper][actr]>=0.197)
                        cls=2;
                    else if (dwddensity[dctr][dper][actr]>=0.14)
                        cls=3;
                    else if (dwddensity[dctr][dper][actr]>0.021 &&
dwdld[dctr][dper][actr][lctr]<15)
                        cls=4;
                    else if (dwddensity[dctr][dper][actr]>0.13 &&
dwdld[dctr][dper][actr][lctr]>=15 &&
                        dwdld[dctr][dper][actr][lctr]<25)
                        cls=4;
                    else if (dwddensity[dctr][dper][actr]>0.021 &&
dwdld[dctr][dper][actr][lctr]>=25)
                        cls=4;
                    else
                        toodead=1;

                    if (dwdln[dctr][dper][actr][lctr]>=16)
                        ln=0;
                    else if ((dwdln[dctr][dper][actr][lctr]<16 &&
dwdln[dctr][dper][actr][lctr]>=8))
                        ln=1;
                    else
                        tooshort=1;

                    if (toosmall==0 && toodead==0 && tooshort==0)
                        reportdwd[ln][diam][cls]+=dwdln[dctr][dper][actr][lctr]
                            *dwdpa[dctr][dper][actr][lctr];

//                  fprintf(editfp,"%4.2f %4.2f %4.2f
%4.2f",dwdld[dctr][dper][actr][lctr],
//                      dwdln[dctr][dper][actr][lctr],dwdpa[dctr][dper][actr][lctr],
//                      dwddensity[dctr][dper][actr][lctr]);

                    toosmall=0;
                    toodead=0;
                    tooshort=0;
                }
            }
        }
    }
  }
}

ECOCLASS
#include "var.h"
extern int retpag[linesor];

void ecoclass(int retpag[], int pagindex)
```

```
{
    int pagctr;
    /* for the purpose of checking verses FVS I changed the first line where pag==1
     The true value is 340, the FVS value for checking is 899*/
    int pagval[8][5]={340,57,1,5,1,
                      571,67,1,5,1,
                      803,78,1,1,1,
                      977,92,1,1,1,
                      1096,87,1,1,1,
                      1414,77,1,9,2,
                      1217,95,0,1,2,
                      1303,76,1,6,1};
    /*   int pagval[8][5]={899,85,1,1,1, // this line for checking FVS
                      571,67,1,5,1,
                      803,78,1,1,1,
                      977,92,1,1,1,
                      1096,87,1,1,1,
                      1414,77,1,9,2,
                      1217,95,0,1,2,
                      1303,76,1,6,1};
    /* 0=PIJE, 1=pine/oak, 2=PSME-DRY, 3=PSME-WET, 4=ABCO-DRY
      5=ABCO-WET (WF, site species), 6-ABCO-WET (DF, non-site species)
      7=ABMA -- [8]
      0=SDI, 1=SI, 2=FLAG (is this the site species?), 3=SP CODE, 4=NUM OF SP IN
PAG */


    for (pagctr=0; pagctr<5; pagctr++)
        retpag[pagctr]=pagval[pagindex][pagctr];


}
FLAMELNFN
#include "var.h"
extern int lines;

double flamelnfn(double dbh[], double tpa[], double cw[], int reportsp[], double ba[])
{
    int vegclass, covertype, qmdint, coverint;
    double flame[9][7][2]={
        6.6, 0, 6.6, 1.2, 5.8, 1.2, 5.8, 1.2, 5.8, 5.8, 5.8, 5.8, 5.8, 5.8,
         24, 0,  24, 6.6, 6.6, 3.4, 6.6, 3.4, 6.6, 3.4, 3.4, 3.4, 3.4, 3.4,
        6.6, 0, 6.6, 3.4, 5.8, 3.4, 5.8, 3.4, 5.8, 3.4, 5.8, 5.8, 5.8, 5.8,
        7.3, 0, 6.6, 3.4, 6.6, 3.4, 6.6, 3.4, 6.6, 3.4, 3.4, 3.4, 3.4, 3.4,
        6.6, 0, 6.6, 1.2, 5.8, 1.2, 5.8, 1.2, 5.8, 5.8, 5.8, 5.8, 5.8, 5.8,
        24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
        7.3, 0, 7.3, 3.4, 7.3, 3.4, 7.3, 3.4, 7.3, 3.4, 3.4, 3.4, 3.4, 3.4,
        7.0, 0, 7.0, 1.2, 5.8, 1.2, 5.8, 1.2, 5.8, 1.2, 1.2, 1.2, 1.2, 1.2,
        7.0, 0, 7.0, 1.2, 5.8, 1.2, 5.8, 1.2, 5.8, 1.2, 1.2, 1.2, 1.2, 1.2};
    double flameln;
    vegclass=vegclassification(dbh,tpa,cw,reportsp,ba);
    covertype=(int)vegclass/100;
    qmdint=(int)(vegclass-covertype*100)/10;
    coverint=(int)(vegclass-covertype*100-qmdint*10);
    covertype-=1;

    flameln=flame[covertype][qmdint][coverint];

    return flameln;
}
FORMCL
#include "var.h"
extern double frmcls[totsp];

double formcl(int sp, double dbh)
{
    double ifcdbh, fc;
    double siskfc[5][totsp]={
        72, 76, 66, 76, 72, 76, 75, 78, 74, 80,
        72, 76, 70, 76, 72, 76, 75, 78, 74, 80,
        72, 74, 70, 76, 72, 76, 75, 78, 74, 78,
        72, 74, 68, 74, 72, 74, 74, 75, 74, 76,
        72, 72, 66, 72, 72, 72, 74, 74, 74, 75};

    if (frmcls[sp]<=0)
    {
        ifcdbh=(dbh-1.0)/10.0;
        if (ifcdbh<0)
            ifcdbh=0;
        if (dbh>40.9)
            ifcdbh=4;

        fc=siskfc[(int)ifcdbh][sp];
    }
    else
    {
        fc=frmcls[sp];
        if (fc<=0)
            fc=80;
    }

    return (fc);
}
GROWT
#include "var.h"
extern int lines, per;
extern int flagforperiod, initialize, baptr[linesor], sp[linesor];
extern double tpa[linesor], dbh[linesor], restpa[linesor], dg[linesor], resdbh[linesor],
htg[linesor], resht[linesor];
extern double rescr[linesor], cr[linesor];
extern double ba[linesor], pba, vol[linesor], futba[linesor];
extern double cwco1[totsp], cwco2[totsp], cwco3[totsp], rescw[linesor];
extern FILE *growfp;
float abirth[linesor], ag;
extern float elev, slope, aspect;

// extern vars for ecoclass
int retpag[linesor];
double si[totsp], siage[linesor];
double sitecomp[totsp], sdi[totsp];
double *dbhfull[linesor];

// calibration and regent
double hcor[totsp], cortem[totsp],snp[totsp],snx[totsp],sny[totsp],hk,dk,dkk;
double cornew, xba, xhtgr, edh, term, xrhgro, xwt, con, dds;
int numcal[linesor];
float fint=5.0;

// coefficients
double lat[totsp]={0,0,0.054,0,0,0,0.1434,-0.4297,0,0};
double d1co1[totsp]={0,0,0,0,0,0,-0.007,0,0,0};
double d1co2[totsp]={0,0,0,0,0,0,-0.834,0,0,0};
double d1co3[totsp]={0,0,0.012,0,0,0,0.00734,0.01401,0,0};
double d1co4[totsp]={0,0,1.41389,0,0,0,1.53339,1.26883,0,0};
double d1co5[totsp]={0,0,-0.48938,0,0,0,-0.47442,-0.35325,0,0};
double d1co6[totsp]={0,0,0.3266,0,0,0,0.35739,0.27986,0,0};
double d1co7[totsp]={-0.16,0,-0.16,-0.32497,-0.79922,-0.32497,-0.44256,-0.79922,0,-
0.44256};
double d1co8[totsp]={-0.25287,-0.24596,-0.25287,-0.20902,0,-0.20902,-0.12359,0,-
0.35579,-0.12359};
double d1co9[totsp]={0,0,0,0.80370,0,0.80370,0,0,0,0};
double d2co1[totsp]={-0.11954,-0.040708,0,0,-0.10656,0,0,0,-0.03587,-0.0156};
double d2co2[totsp]={0.08632,-0.16836,0,0,-0.19174,0,0,0,-0.19935,-0.1563};
double d2co3[totsp]={0.85815,0.46468,0,0,-1.29627,0,0,0,0.73530,0.58937};
double d2co4[totsp]={-1.17209,-0.87145,0,0,0.87335,0,0,0,-0.99561,-1.05045};
double d2co5[totsp]={0.32093,0.56356,0,1.10842,0.20189,1.10842,0,0,0.00659,0.47360};
double d2co6[totsp]={1.23911,0.8699,0,0.96865,1.14082,0.96865,0,0,0.99531,1.01718};
double d2co7[totsp]={-1.20841,2.9604,0,1.5466,2.82796,1.5466,0,0,2.08524,3.01884};
double d2co8[totsp]={2.31782,-1.08219,0,0.07152,-2.14739,0.07152,0,0,-0.98396,-
1.12464};
double d2co9[totsp]={-0.000338,-0.000313,0,-0.000728,-0.000875,-0.000728,0,0,-
0.000373,-0.000356};
double d2co10[totsp]={-0.00199,-0.00443,0,-0.00408,-0.00126,-0.00408,0,0,-0.00147,-
0.00257};
double d2co11[totsp]={0,0,0,-0.00002,0,-0.00002,0,0,-0.00018,0};
double d2co12[totsp]={0,0,0,0,0.56348,0,0,0,0.50155,0};
double d2co13[totsp]={0,0,0,0,0,0,-0.007,0,0,0};
double d2co14[totsp]={0,-0.01744,0,0,0,0,0,0,0,-0.16596};
double
volabhco1[totsp]={0.000887,0.001168,0.000887,0.000887,0.000887,0.001265,0.000887,0.
000866,0.000887,0.00108};
double
volabhco2[totsp]={0.367622,0.265430,0.367622,0.367622,0.367622,0.172813,0.367622,0.
38394,0.367622,0.3583};
double dib1co1[totsp]={0,0,0,0,0,0,0,0,0,0.287414};
double
dib1co2[totsp]={0.989819,0.989819,0.989819,0.989819,0.989819,1,0.989819,1.03908,0.9
89819,0.828652};
double dib1co3[totsp]={1,1,1,1,1,1,1,1,1,1.082631};
double
dibco1[totsp]={0.903563,0.903563,0.903563,0.903563,0.903563,0.809427,0.903563,0.859
045,0.903563,0.904973};
double
dibco2[totsp]={0.989388,0.989388,0.989388,0.989388,0.989388,1.016866,0.989388,0.837
291,0.989388,1};
double htabh,volabh,volbbh,dib, dib1, r;
double k1,k2,k3=0.000162,k4,k5,k6,k7;
double
pccfco1[totsp]={0.0204,0.0388,0.0194,0.0219,0.0212,0.0219,0.0172,0.0392,0.0356,0.069}
;
double
pccfco2[totsp]={0.0123,0.0269,0.0142,0.0169,0.0084,0.0169,0.00876,0.018,0.0136,0.0225
};
double
pccfco3[totsp]={0.0074,0.00466,0.00261,0.00325,0.0033,0.00325,0.00112,0.00207,0.0052
4,0.00183};
double
pccfco4[totsp]={0.009187,0.017299,0.008915,0.007813,0.011109,0.007813,0.011402,0.00
7244,0.007875,0.015248};
double pccfco5[totsp]={1.76,1.5571,1.78,1.778,1.725,1.778,1.756,1.8182,1.736,1.7333};
double loc[totsp]={-2.68349,-2.41928,0,-4.6744,-1.6995,-4.6744,0,0,-0.94563,-2.06853};
double
sdco1[totsp]={4.8042,4.78737,4.89619,4.23251,4.73881,4.23251,4.83642,4.74961,4.6618
1,4.80268};
double sdco2[totsp]={-9.92422,-7.31698,-12.55873,-8.31711,-9.44913,-8.31711,-7.04795,-
7.19103,-8.33117,-8.40657};
double dhtco1[totsp]={4.80758,0,0,0,4.4666,0,0,0,4.9684,0};
double dhtco2[totsp]={-0.00224,0,0,0,-0.00179,0,0,0,-0.004057,0};
double dhtco3[totsp]={-0.000513,0,0,0,0.002048,0,0,0,0.000924,0};
double dhtco4[totsp]={-7.729644,0,0,0,-7.9428,0,0,0,-10.45158,0};
double dshtco1[totsp]={3.817,0,0,0,3.56,0,0,0,3.385,0};
double dshtco2[totsp]={-0.78296,0,0,0,-0.54648,0,0,0,-0.58984,0};
double spsdi[totsp]={382,547,706,571,588,571,800,647,759,759};
double
mcsco1[totsp]={6.82187,7.48846,5.12357,6.04928,3.64292,6.04928,6.14578,6.92893,5.95
912,7.44422};
```

```
double mcsco2[totsp]={-0.02247,-0.02899,-0.01042,-0.01091,-0.00317,-0.01091,-
0.02781,-0.04053,-0.01812,-0.04779};
double
wbco1[totsp]={0.06607,0.52909,0.29964,0.03685,0.08402,0.03685,0.16601,0.25115,0.256
67,0.48464};
double
wbco2[totsp]={1.10705,1.00677,1.05398,1.09499,1.10297,1.09499,1.08150,1.05987,1.064
74,1.01272};
double wcco1[totsp]={2.04714,-3.48211,-
1.0927,4.0134,0.91078,4.01340,0.91420,0.33383,0.11729,-2.78353};
double
wcco2[totsp]={0.15070,1.38780,0.80687,0.04946,0.45819,0.04946,0.45768,0.63833,0.616
81,1.27283};
double waco[totsp]={0,0,0,0,0,0,0,0,0,0};
double
ht1[totsp]={4.80420,4.78737,4.89619,4.23251,4.73881,4.23251,4.83642,4.74961,4.66181,
4.80268};
double ht2[totsp]={-9.92422,-7.31698,-12.55873,-8.31711,-9.44913,-8.31711,-7.04795,-
7.19103,-8.33117,-8.40657};


void growtrees(double restpa[],double dbh[],double ht[])
{
    int gtctr, pbalctr, crctr, rankctr, crflag;
    float elev=45, slope=5, aspect=0;
    int site[totsp]={63,110,84,110,63,110,110,99,63,110};
    double oldht, newht;
    double bal, pccf, hoavh, srelht, ldds, large40, large40tpa, tccf;
    double gtsumdbhtpa, gtsumtpa, gtsdi, relsdi, meancrstand, wa, wb, wc, scale;
    int  dbhfullctr;
    double pct, pcthat, diff, pdiff, crln, crmax, dgdib, dgdibsq;
    float totlines=(float)lines;

// vars for sichg, height
    double xmod=1, relht, cr10;
    int jsisp, isisp, cnteco, pagindex=pag, retctr, agmaxflag, ipassflag;
    int ismall=0, toler=2, ipass=0, htmax=300, agmax=200;
    double hguess, htdiff, hold, brat, dbhbrat;
    float scale3,regyr,finth=5, yr=5, scale2;

// sort array of dbh or ba
    dbhfullctr=0;
    for (gtctr=0; gtctr<lines; gtctr++)
        if (restpa[gtctr]>0)
        {   dbhfull[dbhfullctr]=&dbh[gtctr];
            dbhfullctr++;
        }
    quicksort2(dbhfull,0,dbhfullctr-1);

    gtsumdbhtpa=0;
    gtsumtpa=0;
    for (crctr=0; crctr<lines; crctr++)
    {   gtsumdbhtpa+=pow(dbh[crctr],2)*restpa[crctr];
        gtsumtpa+=restpa[crctr];
    }
    gtsdi=gtsumtpa*pow((pow(gtsumdbhtpa/gtsumtpa,0.5)/10),1.605);


// inititalizing arrays for calibrating small tree height section
    if (initialize==1)
    {
        for (gtctr=0; gtctr<totsp; gtctr++)
        {   hcor[gtctr]=0;
            cortem[gtctr]=0;
            numcal[gtctr]=0;
            snp[gtctr]=0;
            snx[gtctr]=0;
            sny[gtctr]=0;
        }
        scale3=regyr/finth;
        cornew=1;

        jsisp=0;
        isisp=-1;
        cnteco=0;

        for (retctr=0; retctr<totsp; retctr++)
        {   sitecomp[retctr]=0;
            sdi[retctr]=0;
        }
        retctr=pag;

        do
        {   ecoclass(retpag,retctr);                    // sets SDI AND SI
                                                            // for site species
            cnteco+=1;

            if (jsisp==0 && retpag[2]==1)
                jsisp=retpag[3];
            if (isisp<0 && retpag[2]==1)
                isisp=retpag[3];
            if (sitecomp[retpag[3]]<=0)
                sitecomp[retpag[3]]=retpag[1];
            if (sdi[retpag[3]]<=0)
                sdi[retpag[3]]=retpag[0];
            if (retpag[4]>=1 && cnteco<retpag[4]) // for ABCO wet there are two species:
DF & WF
                retctr+=1;
        } while (cnteco<retpag[4]);
        sichg(isisp,sitecomp[isisp],siage);                  // sets siage

        for (retctr=0; retctr<totsp; retctr++)
            si[retctr]=htcalc(sitecomp[isisp],isisp,siage[retctr]); // calc site for all species

        for (retctr=0; retctr<totsp; retctr++)              // only use si where we
        {   if (sitecomp[retctr]==0)                        // don't already have a site
                sitecomp[retctr]=si[retctr];                // sitecomp is final site array
        }

        for (retctr=0; retctr<totsp; retctr++)
        {   if (sdi[retctr]==0)
                sdi[retctr]=sdi[isisp]*spsdi[retctr]/spsdi[isisp];
        }
    }

    if (per==3)
        printf("");
    if(flagforperiod==1)
    {
        quicksort(baptr,0,lines-1,ba);
        flagforperiod=0;
    }

    pbalctr=0;
    pba=0;
    pccf=0;
    large40=0;
    large40tpa=0;
    for(pbalctr=0; pbalctr<lines; pbalctr++)
    {
        pba+=ba[baptr[pbalctr]]*restpa[baptr[pbalctr]];

        if (large40tpa<40)
        {   large40+=ht[baptr[pbalctr]]*restpa[baptr[pbalctr]];
            large40tpa+=restpa[baptr[pbalctr]];
            if (large40tpa>40)
            {
                large40-=ht[baptr[pbalctr]]*(large40tpa-40);
                large40tpa-=(large40tpa-40);
            }
        }

        if (dbh[pbalctr]>=1)
            tccf=pccfco1[sp[pbalctr]] +pccfco2[sp[pbalctr]]*dbh[pbalctr]
            +pccfco3[sp[pbalctr]]*pow(dbh[pbalctr],2);
        else if(dbh[pbalctr]>0.1)
            tccf=pccfco4[sp[pbalctr]]*pow(dbh[pbalctr],pccfco5[sp[pbalctr]]);
        else
            tccf=0.001;
        pccf+=tccf*restpa[pbalctr];
    }

    large40=large40/large40tpa;

// grow
    for (gtctr=0; gtctr<lines; gtctr++)
    {
        if (restpa[gtctr]>0)
        {
            bal=0;
            pbalctr=0;
            while (ba[baptr[pbalctr]]>ba[gtctr])
            {
                bal+=ba[baptr[pbalctr]]*restpa[baptr[pbalctr]];
                pbalctr++;
            }

            hoavh=ht[gtctr]/large40;
            if (hoavh>1.5)
                srelht=1.5;
            else
                srelht=hoavh;

    // diameter growth trees DBH >= 3"

            if(dbh[gtctr]>=0)//3)
            {
                if(sp[gtctr]==2 || sp[gtctr]==6 || sp[gtctr]==7) // dbh
                {   ldds=lat[sp[gtctr]]
                        +d1co1[sp[gtctr]]*elev
                        +d1co9[sp[gtctr]]*(slope/100)
                        +d1co2[sp[gtctr]]*pow((slope/100),2)
                        +d1co3[sp[gtctr]]*sitecomp[sp[gtctr]];
                    ldds+= d1co4[sp[gtctr]]*log(dbh[gtctr])
                        +d1co5[sp[gtctr]]*pow(dbh[gtctr],2)/1000
                        +d1co6[sp[gtctr]]*(pow((cr[gtctr]),2)/log(dbh[gtctr]+1))/1000
                        +d1co7[sp[gtctr]]*bal/(log(dbh[gtctr]+1)*100)
                        +d1co8[sp[gtctr]]*log(pba);

                    ldds=log(exp(ldds)/2);
                }
                else
                {
                    ldds=loc[sp[gtctr]] +d2co1[sp[gtctr]]*sin(aspect)*(slope/100)
                        +d2co13[sp[gtctr]]*(pow(elev,2))
                        +d2co2[sp[gtctr]]*cos(aspect)*(slope/100)
                        +d2co3[sp[gtctr]]*(slope/100)
+d2co4[sp[gtctr]]*pow((slope/100),2)
                        +d2co5[sp[gtctr]]*log(sitecomp[1]);
                    ldds+=d2co6[sp[gtctr]]*log(dbh[gtctr])
                        +d2co7[sp[gtctr]]*(cr[gtctr]/100)
+d2co8[sp[gtctr]]*pow((cr[gtctr]/100),2)
                        +d2co9[sp[gtctr]]*pow(dbh[gtctr],2)
                        +d2co10[sp[gtctr]]*bal/log(dbh[gtctr]+1) +d2co11[sp[gtctr]]*pccf
                        +d2co12[sp[gtctr]]*srelht
                        +d2co14[sp[gtctr]]*log(pba);
                    if (sp[gtctr]==9)
                        ldds-=0.15032;
                }
                if (ldds<-9.21)
                    ldds=-9.21;

                dgdib=dbh[gtctr]*bratio(sp[gtctr],dbh[gtctr]);
                ldds=exp(ldds);
                dgdibsq=pow(dgdib,2);
                dg[gtctr]=(pow((dgdibsq+ldds),0.5)-dgdib);
                brat=bratio(sp[gtctr],dbh[gtctr]);
            }

    // beginning of htgf
            if (sp[gtctr]==1||sp[gtctr]==2||sp[gtctr]==3||sp[gtctr]==5||sp[gtctr]==6||
                sp[gtctr]==7||sp[gtctr]==9)                              // conifer
            {
                if (ht[gtctr]<htmax)
                {
                    if (ipass==0)
                    {
                        ipass=1;
                        ag=abirth[gtctr];

                        if (ag==0)
                            ag=2;

                        agmaxflag=0;
                        ipassflag=0;

                        do
                        {
                            hguess=htcalc(sitecomp[sp[gtctr]],sp[gtctr],ag);

                            if (agmaxflag==1||ipassflag==1)
                                ipass=2;

                            htdiff=fabs(hguess-ht[gtctr]);
                            if ((htdiff<=toler || ht[gtctr]<hguess) && ipass!=2)
                            {
                                abirth[gtctr]=ag;
                                hold=hguess;
                                ag+=5;
                                ipassflag=1;
                            }
                            else if ((htdiff>toler || ht[gtctr]>=hguess) && ipass!=2)
                            {
                                ag+=2;
                                if (ag>agmax)
                                {
                                    hold=hguess;
                                    abirth[gtctr]=ag;
                                    ag-=5;
                                    agmaxflag=1;
                                }
                            }
                        } while (ipass<2);

                    } ipass=0;

                    htdiff=hguess-hold;
                    if (pccf>=50)
                    {
                        relht=ht[gtctr]/large40;
                        if (pccf<100)
                            relht=(relht+1)/2;
                        if (relht>1)
                            relht=1;
                        cr10=cr[gtctr]/10;
                        xmod=-0.02647 +0.71338*pow(relht,2) +0.06851*cr10;
                    }
                    else
                        xmod=1;

                    htg[gtctr]=htdiff*xmod;
                    if (htg[gtctr]<=0)
                        htg[gtctr]=0.001;

                }
                else
                    htg[gtctr]=0;

            }
            else if(sp[gtctr]==0 || sp[gtctr]==4 || sp[gtctr]==8)
            {
                brat=bratio(sp[gtctr],dbh[gtctr]);
                dbhbrat=dbh[gtctr]+dg[gtctr]/brat;
                oldht=exp((dhtco1[sp[gtctr]] +dhtco2[sp[gtctr]]*cr[gtctr]
+dhtco3[sp[gtctr]]*sitecomp[sp[gtctr]])
                        +dhtco4[sp[gtctr]]/(dbh[gtctr]+1)) +4.5;
                newht=exp((dhtco1[sp[gtctr]] +dhtco2[sp[gtctr]]*cr[gtctr]
+dhtco3[sp[gtctr]]*sitecomp[sp[gtctr]])
                        +dhtco4[sp[gtctr]]/(dbhbrat+1)) +4.5;
                htg[gtctr]=newht-oldht;
            }



    // this section is for initializing and only needs to be done once or at the start of each
period.
    // I don't know which
            if (initialize==1)
            {
                initialize=0;
                scale3=1;
                for (crctr=0; crctr<lines; crctr++)
                    numcal[crctr]=0;
                for (crctr=0; crctr<lines; crctr++)
                    if (dbh[crctr]<5)// && ht[crctr]>=0.01 && htg[crctr]>=0.001)
                    {
                        cr10=cr[crctr]/10;
                        relht=ht[crctr]/large40;
                        if (relht>1.5)
                            relht=1.5;
                        xba=ba[crctr];
                        if (xba<=0)
                            xba=0.1;
                        xhtgr=htgr5(sp[crctr],sitecomp[sp[crctr]],xba,relht,cr10,ht[crctr]);

                        edh=xhtgr;
                        term=htg[crctr]*scale3;
                        snp[sp[crctr]]=restpa[crctr];
                        snx[sp[crctr]]=restpa[crctr]*edh;
                        sny[sp[crctr]]=restpa[crctr]*term;
                        numcal[sp[crctr]]+=1;
                    }

                for (crctr=0; crctr<totsp; crctr++)
                {
                    cornew=1;
                    if (numcal[crctr]>=5)
                    {
                        snx[crctr]=snx[crctr]/snp[crctr];
                        sny[crctr]=sny[crctr]/snp[crctr];
                        cornew=sny[crctr]/snx[crctr];
                        if (cornew<=0)
                            cornew=0.0001;
                    }
                    cortem[crctr]=cornew;
                    hcor[crctr]=log(cornew);
                }
            } //end of if statement for initialize
        // end of initializing section
    if (dbh[gtctr]<=5)
    {
        scale=yr/fint;
        scale2=fint/5;
        xrhgro=scale2;
        con=exp(hcor[sp[gtctr]]);

        xba=ba[gtctr];
        if (xba<=0)
            xba=0.1;
        cr10=cr[gtctr]/10;
        relht=ht[gtctr]/large40;
        if (pccf<=75)
            relht=1-((relht-1)/75)*pccf;
        if (relht>1.5)
            relht=1.5;
        xhtgr=htgr5(sp[gtctr],sitecomp[sp[gtctr]],xba,relht,cr10,ht[gtctr]);
        xhtgr*=con*xrhgro;

// weights for trees with DBH <5

        xwt=0;
        if(dbh[gtctr]>2)
            xwt=(dbh[gtctr]-2)/3;
        htg[gtctr]=xhtgr*(1-xwt)+xwt*htg[gtctr];

// calculate dbh growth for trees less than 3" DBH
        if (dbh[gtctr]<3)
        {
            hk=ht[gtctr]+htg[gtctr];
            if (hk<4.5)
            {   dbh[gtctr]+=hk*0.001;
                dg[gtctr]=0;
            }
            else
            {   dk=ht2[sp[gtctr]]/(log(hk-4.5)-ht1[sp[gtctr]])-1;
                if (ht[gtctr]<4.5)
                    dkk=dbh[gtctr];
                else
                    dkk=ht2[sp[gtctr]]/(log(ht[gtctr]-4.5)-ht1[sp[gtctr]])-1;
            }


            dk=htdbh(sp[gtctr],dk,hk,1);

            if (ht[gtctr]<=4.5)
                dkk=dbh[gtctr];
            else
                dkk=htdbh(sp[gtctr],dkk,ht[gtctr],1);
```

```
                    brat=bratio(sp[gtctr],dbh[gtctr]);
                    if (dk<0 || dkk<0)
                    {    dg[gtctr]=htg[gtctr]*0.2*brat;
                         dk=dbh[gtctr]+dg[gtctr];
                    }
                    else
                         dg[gtctr]=(dk-dkk)*brat;
                    if (dg[gtctr]<0)
                         dg[gtctr]=0;

                    dds=dg[gtctr]*(2*brat*dk +dg[gtctr])*scale;
                    dg[gtctr]=pow((pow(dk*brat,2)+dds),0.5)-brat*dk;
               }
          }

          resht[gtctr]=ht[gtctr]+htg[gtctr];
          resdbh[gtctr]=dbh[gtctr]+dg[gtctr]/brat;
          futba[gtctr]=pow((resdbh[gtctr]/24),2)*pi;

          if (ht[gtctr]>4.5)
               rescw[gtctr]=cwco1[sp[gtctr]]*pow(resdbh[gtctr],cwco2[sp[gtctr]]);
          else
               rescw[gtctr]=cwco3[sp[gtctr]]*resht[gtctr];

          htabh=resht[gtctr]-4.5;
          if (htabh>0)

volabh=volabhco1[sp[gtctr]]*(pow(htabh/resdbh[gtctr],volabhco2[sp[gtctr]]))
               *(pow(resdbh[gtctr],2))*htabh;
          else
               volabh=0;

          dib=dibco1[sp[gtctr]]*(pow(resdbh[gtctr],dibco2[sp[gtctr]]));

dib1=dib1co1[sp[gtctr]]+dib1co2[sp[gtctr]]*(pow(resdbh[gtctr],dib1co3[sp[gtctr]]));
          r=pow((dib/dib1),2/3);

          k1=0.25*3.14156*pow(dib1,2);
          k2=(1/43904)*(729+81*r +297*pow(r,2) +265*pow(r,3));
     //   k3=1/6174;
          k4=pow((4.5-r),3);
          k5=1.5*pow((4.5-r),2)*(1-r);
          k6=(4.5-r)*pow((1-r),2);
          k7=pow((1-r),3);

          volbbh=k1*(k2-k3*(k4-k5+k6-k7));
          if (volbbh<0)
               volbbh=0;
          vol[gtctr]=volbbh+volabh;
```

```
// change the crown ratio

            crctr=gtctr;
            relsdi=gtsdi/sdi[sp[crctr]];// spsdi
            if (relsdi>1.5)
                relsdi=1.5;

            meancrstand= mcsco1[sp[crctr]] +mcsco2[sp[crctr]]*relsdi*100;
            wa=waco[sp[crctr]];
            wb=wbco1[sp[crctr]] +wbco2[sp[crctr]]*meancrstand;
            wc=wcco1[sp[crctr]] +wcco2[sp[crctr]]*meancrstand;
            if(wb<3)
                wb=3;
            if(wc<2)
                wc=2;

            scale=1.5-relsdi;
            if(scale>1)
                scale=1;
            if(scale<0.3)
                scale=0.3;

            rankctr=0;
            while (dbh[crctr]>*dbhfull[rankctr])
                rankctr++;
            pct=(rankctr+1)/totlines*scale;
            if (pct<0.05)
                pct=0.05;
            if (pct>0.95)
                pct=0.95;

            crflag=0;
            diff=1;

            pcthat=1-exp(-(pow((meancrstand-wa)/wb,wc)));
            diff=pct-pcthat;

            while (diff!=0)
            {   if (diff<0)
                {   if (crflag==1)
                    {   diff=0;
                        meancrstand-=0.1;
                    }
                    else
                    {   meancrstand-=0.1;
                        crflag=-1;
                        pcthat=1-exp(-(pow((meancrstand-wa)/wb,wc)));
                        diff=pct-pcthat;
                    }
                }
```

```
                    if (diff>0)
                    {   if (crflag==-1)
                        {   diff=0;
                            meancrstand+=0.1;
                        }
                        else
                        {   meancrstand+=0.1;
                            crflag=1;
                            pcthat=1-exp(-(pow((meancrstand-wa)/wb,wc)));
                            diff=pct-pcthat;
                        }
                    }
                } // end of while, adjusting meancrstand

                meancrstand*=10;
                diff=meancrstand-cr[gtctr];
                pdiff=diff/cr[gtctr];
                if (pdiff<=0.1 && pdiff>=-0.1)
                    meancrstand=cr[gtctr]+diff;
                else if (pdiff<-0.1)
                    meancrstand=cr[gtctr]+cr[gtctr]*-0.1;
                else
                    meancrstand=cr[gtctr]+cr[gtctr]*0.1;

                crln=ht[gtctr]*cr[gtctr]/100;
                crmax=(crln+htg[gtctr])/resht[gtctr]*100;
                if (meancrstand>crmax)
                    meancrstand=crmax;
                if (meancrstand>95)
                    meancrstand=95;
                if (meancrstand<10)
                    meancrstand=10;

                rescr[gtctr]=meancrstand;

            } // end of if growtrees
            else
                vol[gtctr]=0;
        }
    }
HTCALC
#include "var.h"

double htcalc(double sindx, int ispc, double ag)
{   double hguess=0, a=0, b=0, c=0, d=0;
    double htcco0[totsp]={0,2500,69.91,0,0.375,0,69.91,0,0.204,69.91};
    double htcco1[totsp]={6.413,-
0.95038,38.0202,1.88,31.233,1.88,38.0202,1.88,39.787,38.0202};
    double htcco2[totsp]={0.322,0.109757,-1.05213,7.178,0,7.178,-1.05213,7.178,0,-
1.05213};
```

```
    double htcco3[totsp]={0,0.055818,0.009557,-0.025,0,-0.025,0.009557,-
0.025,0,0.009557};
    double
htcco4[totsp]={0,0.0079224,101.842894,1.64,0,1.64,101.842894,1.64,0,101.842894};
    double htcco5[totsp]={0,-0.0007338,-0.001442,-0.0007338,0,-0.0007338,-
0.001442,0,0,-0.001442};
    double
htcco6[totsp]={0,0.0001977,1.67259,0.0001977,0,0.0001977,1.679259,0,0,1.679259};
    double htcco7[totsp]={0,0,38.020235,0,0,0,38.020235,0,0,38.020235};
    double htcco8[totsp]={0,0,1.052133,0,0,0,-1.052133,0,0,-1.052133};
    double htcco9[totsp]={0,0,0.009557,0,0,0,0.009577,0,0,0.009557};

    if (ispc==0)
    {   a=pow(ag,0.5)-pow(50,0.5);
        hguess=sindx*(1+htcco2[ispc]*a)-htcco1[ispc]*a;
    }
    else if (ispc==1)
    {   d=htcco0[ispc]/(sindx-4.5);
        a=htcco1[ispc] +htcco2[ispc]*d;
        b=htcco3[ispc] +htcco4[ispc]*d;
        c=htcco5[ispc] +htcco6[ispc]*d;
        hguess=pow(ag,2)/(a+b*ag+c*pow(ag,2)) +4.5;
    }
    else if (ispc==2 || ispc==6 || ispc==9)
    {   a=(htcco1[ispc] *pow(ag,htcco2[ispc]) *exp(htcco3[ispc]*ag));
        b=htcco4[ispc]*(1 -exp(htcco5[ispc]*pow(ag,htcco6[ispc])));
        hguess=(sindx-htcco0[ispc] +a*b)/a +4.5;
    }
    else if (ispc==3 || ispc==5 || ispc==7)
        hguess=(htcco1[ispc]*sindx-htcco2[ispc])
                *pow((1-exp(htcco3[ispc]*ag)),(0.001*sindx+htcco4[ispc]));
    else if (ispc==4)
        hguess=sindx/(htcco0[ispc]+htcco1[ispc]/ag);
    else if (ispc==8)
        hguess=sindx/(htcco0[ispc]+htcco1[ispc]/ag);

    return (hguess);
}
```

**HTDBH**

```
#include "var.h"

double htdbh(int sp, double dbh, double ht, int mode)
{   double hat3;

    double
htdbhco1[totsp]={48.6795,523.0987,1530.33,1348.0419,160.6821,1348.0419,202.886,819.
869,679.1972,604.845};
    double
htdbhco2[totsp]={8.9420,5.7243,7.0811,7.0463,4.1677,7.0463,8.7469,6.4531,5.5698,5.983
5};
```

```
    double htdbhco3[totsp]={-1.4832,-0.4109,-0.2544,-0.3076,-0.4954,-0.3076,-0.8317,-
0.3434,-0.3074,-0.3789};

    sp=0;
    dbh=2;
    ht=0;
    if (mode==0)
        ht=0;
    else
        dbh=0;

    if(mode==0)
    {   if (dbh>=3)
            ht=4.5 +htdbhco1[sp]*exp(-1*htdbhco2[sp]*pow(dbh,htdbhco3[sp]));
        else
            ht=((4.5+htdbhco1[sp]*exp(-1*htdbhco2[sp]*pow(3,htdbhco3[sp]))-
4.51)*(dbh-3)/2.7)+4.51;
    }
    else
    {   hat3=4.5 +htdbhco1[sp]*exp(-1*htdbhco2[sp]*pow(3,htdbhco3[sp]));
        if (ht>=hat3)
            dbh=exp(log((log(ht-4.5)-log(htdbhco1[sp]))/(-
1*htdbhco2[sp]))*1/htdbhco3[sp]);
        else
            dbh=(((ht-4.51)*2.7)/(4.5+htdbhco1[sp]*exp(-
1*htdbhco2[sp]*pow(3,htdbhco3[sp]))-4.51))+0.3;
    }

    if (mode==0)
        return (ht);
    else
        return (dbh);
}
HTGR5
#include "var.h"

double htgr5(int sp, double site, double ba, double relht, double cr, double ht)
{   double yhtg;

    double hrelht=4.292, hcrsq=0.0566, hht=0.1699, hsite=0.00768;
    double htgrco1[totsp]={-0.78296,-0.00828,-0.00828,-0.00828,-0.54648,-0.00828,-
0.00828,-0.00828,-0.58984,-0.00828};
    double htgrco2[totsp]={3.817,-2.193,-2.193,-2.193,3.56,-2.193,-2.193,-2.193,3.385,-
2.193};

    sp=0;
    ba=0.1;
    if (sp==0||sp==4||sp==8)
    {   if (ba<5)
            ba=5;
        yhtg=exp(htgrco2[sp]+htgrco1[sp]*log(ba));
    }
    else
        yhtg=htgrco2[sp] +relht*hrelht +hcrsq*pow(cr,2) +hht*ht +htgrco1[sp]*ba
+hsite*site;

    if (yhtg<=0)
        yhtg=0.01;

    return (yhtg);
}
OBJ
#include "var.h"
extern int lines;
extern float slope;
extern double snagscost, bfrev[linesor], bfv[linesor];



double objfunction(double objtpa[],double objdbh[], double vol[])
{   double sumdbhtpa=0, sumtpa=0, meandbh, sumvol=0, modmcf; // sumvol is mcf
    double rev, cost, bfvol;
    double slopeofvalue;
    int ofctr, mdbh=0, mcf=0;

    double groundlc[25][6]=
    { 2829, 2624, 2468, 2346, 2296, 2228,
      2829, 2624, 2468, 2346, 2296, 2228,
      2829, 2624, 2468, 2346, 2296, 2228,
      2829, 2624, 2468, 2346, 2296, 2228,
      1880, 1718, 1606, 1527, 1494, 1452,
      1542, 1395, 1298, 1235, 1208, 1175,
      1406, 1265, 1175, 1117, 1093, 1064,
      1331, 1194, 1107, 1053, 1030, 1003,
      1284, 1149, 1064, 1012,  990,  964,
      1250, 1116, 1033,  983,  961,  936,
      1224, 1092, 1010,  960,  940,  915,
      1204, 1072,  991,  943,  922,  898,
      1187, 1056,  976,  928,  908,  885,
      1172, 1042,  963,  916,  896,  873,
      1160, 1031,  951,  905,  885,  863,
      1149, 1020,  941,  895,  876,  853,
      1139, 1010,  932,  887,  867,  845,
      1135, 1007,  929,  883,  864,  842,
      1133, 1005,  927,  882,  862,  840,
      1130, 1002,  924,  879,  860,  838,
      1128, 1000,  922,  877,  858,  836,
      1127,  999,  921,  877,  857,  836,
      1127,  999,  921,  877,  857,  836,
      1126,  999,  921,  876,  857,  835,
 1127,  999,  921,  876,  857,  836};

double cablelc[25][6]=
{ 2180, 1904, 1757, 1669, 1639, 1602,
  2180, 1904, 1757, 1669, 1639, 1602,
  2180, 1904, 1757, 1669, 1639, 1602,
  2180, 1904, 1757, 1669, 1639, 1602,
  2180, 1904, 1757, 1669, 1639, 1602,
  2180, 1904, 1757, 1669, 1639, 1602,
  2180, 1904, 1757, 1669, 1639, 1602,
  2073, 1799, 1655, 1572, 1542, 1507,
  1983, 1711, 1568, 1489, 1460, 1428,
  1905, 1635, 1493, 1418, 1390, 1359,
  1837, 1568, 1428, 1356, 1328, 1299,
  1777, 1509, 1370, 1301, 1274, 1247,
  1723, 1457, 1319, 1253, 1226, 1200,
  1675, 1410, 1274, 1209, 1183, 1158,
  1633, 1369, 1233, 1171, 1145, 1121,
  1596, 1332, 1197, 1137, 1111, 1088,
  1563, 1300, 1166, 1107, 1082, 1060,
  1534, 1272, 1138, 1081, 1056, 1035,
  1510, 1248, 1115, 1059, 1034, 1013,
  1489, 1228, 1096, 1040, 1015,  995,
  1473, 1212, 1080, 1025, 1001,  981,
  1460, 1200, 1067, 1013,  989,  970,
  1451, 1191, 1059, 1005,  981,  961,
  1446, 1185, 1053, 1000,  976,  957,
  1444, 1184, 1051,  998,  974,  955};

double logprice[57][10]=
{  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0, 269, 500, 345,   0, 345, 329, 363,   0, 329,
   0, 269, 477, 345,   0, 345, 329, 363,   0, 329,
   0, 269, 454, 345,   0, 345, 329, 363,   0, 329,
   0, 269, 431, 345,   0, 345, 329, 363,   0, 329,
   0, 269, 408, 345,   0, 345, 329, 363,   0, 329,
   0, 269, 385, 345,   0, 345, 329, 363,   0, 329,
   0, 269, 379, 345,   0, 345, 329, 363,   0, 329,
   0, 284, 373, 345,   0, 345, 331, 361,   0, 331,
   0, 302, 367, 344,   0, 344, 333, 360,   0, 333,
   0, 317, 361, 344,   0, 344, 330, 360,   0, 330,
   0, 331, 358, 345,   0, 345, 328, 360,   0, 328,
   0, 343, 357, 348,   0, 348, 318, 363,   0, 318,
   0, 356, 357, 352,   0, 352, 307, 366,   0, 307,
   0, 366, 357, 355,   0, 355, 297, 368,   0, 297,
        0, 375, 361, 357,  0, 357, 287, 370,  0, 287,
        0, 381, 365, 360,  0, 360, 281, 373,  0, 281,
        0, 388, 369, 363,  0, 363, 275, 375,  0, 275,
        0, 392, 373, 366,  0, 366, 273, 378,  0, 273,
        0, 396, 378, 370,  0, 370, 271, 382,  0, 271,
        0, 396, 383, 374,  0, 374, 261, 385,  0, 261,
        0, 396, 388, 378,  0, 378, 252, 389,  0, 252,
        0, 393, 394, 382,  0, 382, 255, 393,  0, 255,
        0, 391, 399, 386,  0, 386, 257, 397,  0, 257,
        0, 385, 404, 390,  0, 390, 252, 400,  0, 252,
        0, 378, 409, 394,  0, 394, 248, 404,  0, 248,
        0, 368, 414, 398,  0, 398, 247, 409,  0, 247,
        0, 357, 419, 402,  0, 402, 246, 413,  0, 246,
        0, 343, 422, 406,  0, 406, 246, 417,  0, 246,
        0, 330, 422, 410,  0, 410, 246, 421,  0, 246,
        0, 312, 422, 415,  0, 415. 246, 426,  0, 246,
        0, 293, 422, 421,  0, 421, 246, 432,  0, 246,
        0, 293, 422, 428,  0, 428, 245, 439,  0, 245,
        0, 293, 422, 435,  0, 435, 245, 447,  0, 245,
        0, 293, 422, 444,  0, 444, 245, 456,  0, 245,
        0, 293, 422, 453,  0, 453, 245, 465,  0, 245,
        0, 293, 422, 463,  0, 463, 245, 475,  0, 245,
        0, 293, 422, 472,  0, 472, 245, 485,  0, 245,
        0, 293, 422, 482,  0, 482, 245, 495,  0, 245,
        0, 293, 422, 492,  0, 492, 245, 405,  0, 245,
        0, 293, 422, 501,  0, 501, 245, 515,  0, 245,
        0, 293, 422, 510,  0, 510, 245, 525,  0, 245,
        0, 293, 422, 520,  0, 520, 245, 535,  0, 245,
        0, 293, 422, 530,  0, 530, 245, 545,  0, 245,
        0, 293, 422, 540,  0, 540, 245, 555,  0, 245,
        0, 293, 422, 550,  0, 550, 245, 565,  0, 245};

    rev=0;
    bfvol=0;

    for (ofctr=0; ofctr<lines; ofctr++)
    {   sumdbhtpa+=objdbh[ofctr]*objtpa[ofctr];
        sumvol+=vol[ofctr]*objtpa[ofctr];
        sumtpa+=objtpa[ofctr];
//      fprintf(growfp,"objtpa=%.2f vol=%.2f
sumvol=%.2f\n",objtpa[ofctr],vol[ofctr],sumvol);
        rev+=bfrev[ofctr]*objtpa[ofctr];
        bfvol+=bfv[ofctr]*objtpa[ofctr];
    }

    if (sumtpa==0 && sumvol==0)
        rev=0;
    else
    {
        mdbh=0;
```

```
            meandbh=sumdbhtpa/sumtpa;
//          moddbh=fmod(meandbh,(int)meandbh);
//          if (moddbh>0.5)
//              mdbh=1;
            mdbh=(int)(meandbh+0.5);
            if (mdbh>24)
                mdbh=24;

            sumvol=sumvol/1000;
            modmcf=fmod(sumvol,(int)sumvol);
            if (modmcf>0.5)
                mcf=1;
            mcf+=(int)sumvol;
            if (mcf>5)
                mcf=5;

//          fprintf(growfp,"\nmeandbh=%.2f tpa*dbh=%.2f sumtpa=%.2f
sumvol=%.2f\n",meandbh,sumdbhtpa,sumtpa,sumvol);

            if (slope<40)
                cost=sumvol*groundlc[mdbh][mcf];
            else
                cost=sumvol*cablelc[mdbh][mcf];

    //      cost=sumvol/1000*(costco1*pow(sumvol,costco2));

            slopeofvalue=70.81;
    //      rev=sumvol*(9.91+slopeofvalue*meandbh)-cost-snagscost;
            rev=rev-cost-snagscost;

//          fprintf(growfp," cost=%.2f rev=%.2f\n",cost,rev);
    }

    return (rev);
}
PART
#include "var.h"

int partition(int x[],int first, int last, double ba[])
{   int pivot, i, temp;
    double pivotvalue;

    pivot=first;
    pivotvalue=ba[x[first]];

    for (i=first; i<=last; ++i)
    {   if (ba[x[i]]>pivotvalue)
        {   ++pivot;
            if(i!=pivot)
            {   temp=x[pivot];
```

```
                    x[pivot]=x[i];
                    x[i]=temp;
                }
            }
        }

        temp=x[pivot];
        x[pivot]=x[first];
        x[first]=temp;

        return (pivot);

}

int partition4(int x[][linesor],int first, int last, double ba[], int pf)
{
        int pivot, i, temp;
        double pivotvalue;

        pivot=first;
        pivotvalue=ba[x[pf][first]];

        for (i=first; i<=last; ++i)
        {   if (ba[x[pf][i]]<pivotvalue)
            {   ++pivot;
                if(i!=pivot)
                {   temp=x[pf][pivot];
                    x[pf][pivot]=x[pf][i];
                    x[pf][i]=temp;
                }
            }
        }

        temp=x[pf][pivot];
        x[pf][pivot]=x[pf][first];
        x[pf][first]=temp;

        return (pivot);

}
PART2
#include "var.h"

int partition2(double *x[],int first, int last)
{       int pivot, i;
        //float
        double pivotvalue, *temp;

        pivot=first;
        pivotvalue=*x[first];
```

```
    for (i=first; i<=last; ++i)
    {   if (*x[i]<pivotvalue)
        {   ++pivot;
            if(i!=pivot)
            {   temp=x[pivot];
                x[pivot]=x[i];
                x[i]=temp;
            }
        }
    }

    temp=x[pivot];
    x[pivot]=x[first];
    x[first]=temp;

    return (pivot);

}
PART3
#include "var.h"

int partition3(int x[],int first, int last, double ht[])
{   int pivot, i, temp;
    //float
    double pivotvalue;

    pivot=first;
    pivotvalue=ht[x[first]];

    for (i=first; i<=last; ++i)
    {   if (ht[x[i]]>pivotvalue)
        {   ++pivot;
            if(i!=pivot)
            {   temp=x[pivot];
                x[pivot]=x[i];
                x[i]=temp;
            }
        }
    }

    temp=x[pivot];
    x[pivot]=x[first];
    x[first]=temp;

    return (pivot);

}
QS
#include "var.h"
```

```
extern int baptr[linesor];
extern int htlcptr[linesor];

void quicksort(int baptr[], int first, int last, double ba[])
{    int pivot;

     if (first<last)
     {    pivot=partition(baptr,first,last,ba);

          quicksort(baptr,first,pivot-1,ba);
          quicksort(baptr,pivot+1,last,ba);

     }
}

void quicksort4(int htlcptr[][linesor], int first, int last, double htlc[], int pf)
{
     int pivot;

     if (first<last)
     {
          pivot=partition4(htlcptr,first,last,htlc,pf);

          quicksort4(htlcptr,first,pivot-1,htlc,pf);
          quicksort4(htlcptr,pivot+1,last,htlc,pf);

     }
}
QS2
#include "var.h"
extern float *dbhfull[linesor];

void quicksort2(double *dbhfull[], int first, int last)
{    int pivot;

     if (first<last)
     {    pivot=partition2(dbhfull,first,last);

          quicksort2(dbhfull,first,pivot-1);
          quicksort2(dbhfull,pivot+1,last);

     }
}
QS3
#include "var.h"
extern int htorder[linesor];

void quicksort3(int htorder[], int first, int last, double ht[])
{    int pivot;
```

```
    if (first<last)
    {   pivot=partition3(htorder,first,last,ht);

        quicksort3(htorder,first,pivot-1,ht);
        quicksort3(htorder,pivot+1,last,ht);

    }
}
#include "var.h"
extern int lines, sp[linesor], reportsp[linesor];
extern double ht[linesor], dbh[linesor], cr[linesor], cw[linesor], tpa[linesor];
extern double maxhrvtpa[linesor], ba[linesor];

REGEN
void regen(double rsavetpa[],double rsaveba[], int rsavesp[], double rsavemaxhrvtpa[],
        int rsavelines)
{
    int tolerance[totsp][totpags]={
        3, 3, 3, 3, 3, 3, 3,        // 0=tolerant
        0, 0, 0, 0, 1, 2, 2,        // 1=moderates
        0, 0, 0, 0, 0, 1, 1,        // 2=intolerant
        2, 2, 2, 2, 2, 2, 2,        // 3=sprouters
        3, 3, 3, 3, 3, 3, 3,
        2, 2, 2, 2, 2, 2, 2,
        0, 0, 0, 0, 0, 0, 0,
        1, 1, 1, 1, 1, 1, 1,
        3, 3, 3, 3, 3, 3, 3,
        0, 0, 0, 0, 0, 0, 0};

    double regdbh[totsp]={2.0,0.1,0.1,0.1,1.8,0.1,0.1,0.1,1.3,0.1};
    double reght[totsp]={10.0,3.3,3.3,3.4,12.2,3.4,3.3,3.4,9.4,3.3};
    double regcr[totsp]={80,80,80,80,80,80,80,80,80,80};
    double regcw[totsp]={4.6,1.7,1.4,1.4,9.7,1.4,1.1,1.3,8.6,1.5};

    int ctr, placectr, newtreeflag=0;
    int empty[linesor];
    double pba=0, tpaadd, sprouters=0, preptpa=0, modtpa=0;
    double prespeciestpa[totsp], toltpa[4]={0,0,0,0}, totaltoltpa[4], pct[totsp];

    for (ctr=0; ctr<totsp; ctr++)
        prespeciestpa[ctr]=0;

    for (ctr=0; ctr<lines; ctr++)
    {
        empty[ctr]=0;
        if (tpa[ctr]-maxhrvtpa[ctr]==0)
            empty[ctr]=1;
    }

    for (ctr=0; ctr<rsavelines; ctr++)
```

```
{
    pba+=rsaveba[ctr]*(rsavetpa[ctr]-rsavemaxhrvtpa[ctr]);
    preptpa+=(rsavetpa[ctr]);
    prespeciestpa[rsavesp[ctr]]+=rsavetpa[ctr];
    toltpa[tolerance[rsavesp[ctr]][pag]]+=rsavetpa[ctr];

    if (tolerance[rsavesp[ctr]][pag]==3)
        sprouters+=rsavemaxhrvtpa[ctr];
    else if (tolerance[rsavesp[ctr]][pag]==1)
        modtpa+=rsavetpa[ctr];
}


tpaadd=625 * exp(-.02*pba);

if (tpaadd>=1)
{
    placectr=0;
    while (empty[placectr]==0 && placectr<lines)
        placectr++;

    for (ctr=0; ctr<totsp; ctr++)
    {
        if (toltpa[tolerance[ctr][pag]]>0)
            pct[ctr]=prespeciestpa[ctr]/toltpa[tolerance[ctr][pag]];
        else
            pct[ctr]=0;
    }

    if (sprouters>tpaadd)
    {
        sprouters=tpaadd;

        for (ctr=0; ctr<totsp; ctr++)
        {
            if (ctr==0 || ctr==4 || ctr==8)
            {
                if (prespeciestpa[ctr]>0)
                {
                    tpa[placectr]=sprouters*pct[ctr];
                    maxhrvtpa[placectr]=0;
                // restpa?
                    sp[placectr]=ctr;
                    reportsp[placectr]=ctr;
                    dbh[placectr]=regdbh[ctr];
                    ht[placectr]=reght[ctr];
                    cr[placectr]=regcr[ctr];
                    cw[placectr]=regcw[ctr];
                    ba[placectr]=pow(dbh[placectr]/2,2)*pi;
```

```
                        placectr++;
                        while (empty[placectr]==0 && placectr<lines)
                            placectr++;
                    }
                }
            }
        }
        else
        {
            totaltoltpa[3]=sprouters;
            if (preptpa>0)
                totaltoltpa[1]=(tpaadd-sprouters)*modtpa/preptpa;
            else
                totaltoltpa[1]=0;
            totaltoltpa[0]=(0.000004*pow(pba,3) -0.0034*pow(pba,2) +0.9667*pba
+8.9377)/100;
            if (totaltoltpa[0]>1)
                totaltoltpa[0]=1;
            totaltoltpa[0]*=(tpaadd-sprouters-totaltoltpa[1]);
            totaltoltpa[2]=tpaadd-totaltoltpa[0]-totaltoltpa[1]-totaltoltpa[3];

            for (ctr=0; ctr<totsp; ctr++)
            {
                if (prespeciestpa[ctr]>0)
                {
                    tpa[placectr]=totaltoltpa[tolerance[ctr][pag]]*pct[ctr];
                    maxhrvtpa[placectr]=0;
                    sp[placectr]=ctr;
                    reportsp[placectr]=ctr;
                    dbh[placectr]=regdbh[ctr];
                    ht[placectr]=reght[ctr];
                    cr[placectr]=regcr[ctr];
                    cw[placectr]=regcw[ctr];
                    ba[placectr]=pow(dbh[placectr]/2,2)*pi;

                    placectr++;
                    while (empty[placectr]==0 && placectr<lines)
                        placectr++;
                }
            }

            if (placectr>lines)
                lines=placectr;
        }
    }
}
ROOTDISEASE
#include "var.h"
extern int lines;
extern double tpa[linesor], morttpa[linesor];
```

```
// this is only called if pag==4 || 5 || 6

void rootdisease(double vol[], int sp[])
{
    int ctr;
    double standvol=0, mortvol;
    double dfvol=0, rfvol=0, wfvol=0, dfmort, rfmort=0, wfmort, dfpct=0, rfpct=0,
wfpct=0;

    for (ctr=0; ctr<lines; ctr++)
    {
        if (vol[ctr]>0)
        {
            standvol+=vol[ctr]*tpa[ctr];
            if (sp[ctr]==1)
                dfvol+=vol[ctr]*tpa[ctr];
            else if (sp[ctr]==6)
                rfvol+=vol[ctr]*tpa[ctr];
            else if (sp[ctr]==9)
                wfvol+=vol[ctr]*tpa[ctr];
        }
    }

    mortvol=standvol*0.025;

    if (pag==4 || pag==5)
    {
        wfmort=mortvol*0.65;
        dfmort=mortvol-wfmort;
    }
    else if (pag==6)
    {
        wfmort=mortvol*0.5;
        rfmort=mortvol*0.3;
        dfmort=mortvol-wfmort-rfmort;
    }

    if (dfmort>dfvol)
        dfmort=dfvol;
    if (dfvol>0)
        dfpct=dfmort/dfvol;

    if (rfmort>rfvol)
        rfmort=rfvol;
    if (rfvol>0)
        rfpct=rfmort/rfvol;

    if (wfmort>wfvol)
        wfmort=wfvol;
    if (wfvol>0)
        wfpct=wfmort/wfvol;

    for (ctr=0; ctr<lines; ctr++)
    {
        if (vol[ctr]>0)
        {
            if (sp[ctr]==1)
            {
                tpa[ctr]-=tpa[ctr]*dfpct;
                morttpa[ctr]+=tpa[ctr]*dfpct;
            }
            else if (sp[ctr]==6)
            {
                tpa[ctr]-=tpa[ctr]*rfpct;
                morttpa[ctr]+=tpa[ctr]*rfpct;
            }
            else if (sp[ctr]==9)
            {
                tpa[ctr]-=tpa[ctr]*wfpct;
                morttpa[ctr]+=tpa[ctr]*wfpct;
            }
        }
    }
}
RXDIBS
#include "var.h"
extern double smdia[20], gmogln[20];
extern int numlog;

void rxdibs( double dbh, double fc, double ht, double topd)
{
    double d16, usht, stmfnd, stmcal, stmrat, xd, xm;

    for (numlog=0; numlog<20; numlog++)
    {
        smdia[numlog]=0;
        glogln[numlog]=0;
    }

    numlog=0;
    d16=dbh*(fc/100);

    if (d16<=topd)
    {
        numlog=0;
        smdia[0]=topd;
        glogln[0]=(((dbh*dbh-topd*topd)/(dbh*dbh-d16*d16))*12.3)+4.0;
        if (glogln[0]<0.1)
            glogln[0]=0.1;
        }
        else
        {
            smdia[numlog]=d16;
            glogln[numlog]=16.3;
            usht=ht-16.3;

            do
            {
                numlog++;
                stmfnd=16.3*numlog;
                stmcal=usht-stmfnd;

                if (stmcal>=0)
                {
                    stmrat=stmcal/usht;
                    smdia[numlog]=(stmrat/(0.49*stmrat+0.51))*d16;
                    glogln[numlog]=16.3;
                }
            } while (stmcal>=0 && smdia[numlog]>topd);

            smdia[numlog]=topd;
            usht=ht-16.3;
            xd=topd/smdia[0];
            xm=(xd*0.51*usht)/(1.0-(0.49*xd));
            glogln[numlog]=(usht-xm)-((numlog-1)*16.3);
            if (glogln[numlog]<=0)
                glogln[numlog]=0.1;
        }
    }
    SCALEF
    #include "var.h"

    double scalef(double smd, double xlog)
    {
        int ismd;
        double modsmd;
        double factor;
        double factb[120]={
            0,0.143,0.39,0.676,1.07,0,0,
            0,0,0,0,4.9,6.043,7.14,
            8.88,10,11.528,13.29,14.99,17.499,18.99,
            20.88,23.51,25.218,28.677,31.249,34.22,36.376,
            38.04,41.06,44.376,45.975,48.99,50,54.688,57.66,
            64.319,66.731,70,75.24,79.48,83.91,87.19,92.501,94.99,
            99.075,103.501,107.97,112.292,116.99,121.65,126.525,
            131.51,136.51,141.61,146.912,152.21,157.71,163.288,
            168.99,174.85,180.749,186.623,193.17,199.12,205.685,
            211.81,218.501,225.685,232.499,239.317,246.615,254.04,
            261.525,269.04,276.63,284.26,292.501,300.655,308.97,
        317.36,325.79,334.217,343.29,350.785,359.12,368.38,
        376.61,385.135,393.38,402.499,410.834,419.166,428.38,
        437.499,446.565,455.01,464.15,473.43,482.49,491.7,
        501.7,511.7,521.7,531.7,541.7,552.499,562.501,
        573.35,583.35,594.15,604.17,615.01,625.89,636.66,
        648.38,660,671.7,683.33,695.011};
    double factb1[6]={1.16,1.4,1.501,2.084,3.126,3.749};
    double factb2[6]={1.249,1.608,1.854,2.41,3.542,4.167};
    double factb3[6]={1.57,1.8,2.2,2.9,3.815,4.499};

    ismd=0;
    modsmd=fmod(smd,(int)smd);
    if (modsmd>0.5)
        ismd=1;
    ismd+=(int)smd;
    if (ismd<1)
        ismd=1;
    if (ismd>120)
        ismd=120;
    if (xlog>40)
        xlog=40;

    if (ismd>=6 && ismd <=11)
    {
        ismd-=5;
        if (xlog<=15)
            factor=factb1[ismd];
        else if (xlog<=31)
            factor=factb2[ismd];
        else
            factor=factb3[ismd];
    }
    else
        factor=factb[ismd];

    return (factor);

}
SICHG
#include "var.h"
extern double siage[linesor];

void sichg(int currsp, double ssite, double siage[])
{
    int pnotp[totsp]={0,0,0,1,0,1,0,1,0,0};                    // is the tree a pine?
    int sichgctr, sichdiff;
    int simin[totsp]={30,50,30,40,50,40,30,40,50,30};
    int simax[totsp]={70,150,130,120,100,120,130,120,90,130};
    double ageco1[totsp]={6.0,10.0,10.0,12.0,3.0,12.0,10.0,12.0,4.0,10.0};
    double ageco2[totsp]={-0.05,-0.08,-0.05,-0.05,-0.02,-0.05,-0.06,-0.05,-0.03,-0.07};
    double spread, age2dbh;

    for (sichgctr=0; sichgctr<totsp; sichgctr++)
    {   if (pnotp[currsp]==1 && pnotp[sichgctr]==0)
            sichdiff=-1;
        else if (pnotp[currsp]==pnotp[sichgctr])
            sichdiff=0;
        else if (pnotp[currsp]==0 && pnotp[sichgctr]==1)
            sichdiff=1;

        age2dbh=0;

        if (sichdiff!=0)                                    // if one of the trees
        {   if (ssite<simin[currsp])                        // is a pine and the other isn't
                ssite=simin[currsp];
            else if (ssite>simax[currsp])
                ssite=simax[currsp];

            spread=simax[currsp]-simin[currsp];
            spread=100*(ssite-simin[currsp])/spread;
            age2dbh=ageco1[sichgctr] +ageco2[sichgctr]*spread;
        }

        siage[sichgctr]=50 +age2dbh*sichdiff;

    }
}
STANDHTLC
#include "var.h"
extern int lines, flagforhtlc;
int htlcptr[2][linesor];

double standhtlcfn(double ht[], double tpa[], double cr[], double flameln, int pf, int
signaltoprint)
{
    int sctr, tpa50, htlcflag=0, htlctree;
    int htlcgroup[100];//htlcptr[linesor],
    double htlc[linesor], cfl;
    double standhtlc, sumtpa=0, standtpa=0, threshold50, midstorythreshold, midstorytpa;
    double group, groupint, standhtlcmeters, torching=0;

    for (sctr=0; sctr<lines; sctr++)
    {
        htlc[sctr]=ht[sctr]-ht[sctr]*cr[sctr]/100;
        standtpa+=tpa[sctr];

    }

    if (flagforhtlc==1 || signaltoprint==1)
    {
            for (sctr=0; sctr<lines; sctr++)
                htlcptr[pf][sctr]=sctr;

            quicksort4(htlcptr,0,lines-1,htlc,pf);

            if (pf==1)
                flagforhtlc=0;

        }

        threshold50=standtpa*0.1;
        if (threshold50>50)
            threshold50=50;
        midstorythreshold=standtpa*0.05;
        if (midstorythreshold>5)
            midstorythreshold=5;

        for (sctr=0; sctr<lines; sctr++)
        {
            htlcgroup[sctr]=0;
            group=htlc[sctr]*7.62/25;
            groupint=fmod(group,(int)group);
            if (groupint>0.5)
                htlcgroup[sctr]=1;
            htlcgroup[sctr]+=(int)group;
        }

        sctr=0;
        while (sumtpa<threshold50)
        {
            sumtpa+=tpa[htlcptr[pf][sctr]];
            if (sumtpa<threshold50)
                sctr++;
        }
        tpa50=htlcgroup[htlcptr[pf][sctr]];
        htlctree=htlcptr[pf][sctr];

        while (htlcflag==0)
        {
            midstorytpa=0;
            for (sctr=0; sctr<lines; sctr++)
            {
                if (htlcgroup[sctr]==tpa50+1 || htlcgroup[sctr]==tpa50+2 ||
htlcgroup[sctr]==tpa50+3)
                    midstorytpa+=tpa[sctr];
            }

            if (midstorytpa>midstorythreshold)
            {
                htlcflag=1;
            standhtlc=htlc[htlctree];
        }
        else
        {
            sctr=0;
            tpa50++;
            while (htlc[htlcptr[pf][sctr]]<tpa50*25/7.62 && sctr<(lines-1))
                    sctr++;
            // or if sctr<lines
            // if (sctr==lines)
            //      htlctree=lines-1;
            // set standhtlc=100?
            htlctree=htlcptr[pf][sctr];
        }

        if (tpa50>htlcgroup[htlcptr[pf][lines-1]])
        {
            htlcflag=1;
            standhtlc=htlc[htlctree];
        }

    }


    standhtlcmeters=standhtlc*7.62/25;
    cfl=0.45*pow(0.289*pow(28*standhtlcmeters,1.5),0.46)/3.3;

    // if flameln comes in feet, don't use meters here.

    if (flameln<cfl)
        torching=0;
    else
        torching=1.0;

    return (torching);
}
SUMDEV
#include "var.h"
extern int lines, per, dwdlines, gcctr;
extern float fishtreesgoal, ccgoal;
extern double verticalvargoal, snagsgoal, dwdgoal, fbigoal, feigoal;
extern double reportsnag[5][5], reportdwd[2][5][5], sdensity[linesor];
extern double dwddensity[linesor][periods][periods],dwdld[linesor][periods][periods][2];
extern double dwdln[linesor][periods][periods][2], dwdpa[linesor][periods][periods][2];
extern double cover;
extern double tinsectdev, tfishtreesdev, tccdev, tfbi, tfei, tverticaldev, tsnagsdev, tdwddev;
extern float gc[goalcombos];
extern int ksnag, kstree, kdwd;
```

```
// note: eventhough same names are used (ba, dbh...), these are internal variables and can
represent
// either present of future, depending on which variable is passed
double sumdevfn(int sp[], double ba[], double pba, double cw[], double dbh[], double
restpa[],
                double morttpa[],int pf,double objective, double ht[], double cr[], double
flameln,
                FILE *editfp)
{
    int sdctr, dftrigger=0, wftrigger=0, ptrigger=0, a, b;


    int covertype,qmdint,coverint, signaltoprint=0;
    int vegclass, diam, basp, bafl;
    double dfbadev=0, wfbadev=0, pbadev=0, sumdev=0, insectdev;//, pnvdev=0;
    double dfpba=0, wfpba=0, ppba=0, otpba=0;
    double dfbat[7]={80,80,120,250,250,999999,999999};
    double wfbat[7]={0,0,0,0,0,250,250};
    double pbat[7]={80,80,120,180,120,180,180};
    double fishtrees, fishtreesdev=0, ccdev=0, snags, snagsdev=0, firedev=0;// pnvdev=0;
    double fbi, fei;
    double snagtrees=0, snagtreesdev=0;
    double verticalvar, wildlifedev=0, dwdlinear, dwddev=0;
    double torching, bamortpct=0, totaltrees=0;

    double
cbd[9][7][2]={0.12,0.00,0.12,0.20,0.12,0.21,0.13,0.25,0.17,0.25,0.25,0.00,0.25,0.00,

0.15,0.00,0.15,0.25,0.15,0.25,0.11,0.20,0.13,0.20,0.20,0.00,0.20,0.00,

0.15,0.00,0.15,0.25,0.15,0.25,0.11,0.20,0.13,0.20,0.20,0.00,0.20,0.00,

0.09,0.00,0.09,0.15,0.09,0.17,0.11,0.20,0.13,0.20,0.20,0.00,0.20,0.00,

0.10,0.00,0.10,0.20,0.10,0.25,0.15,0.25,0.15,0.25,0.25,0.00,0.25,0.00,

0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,

0.09,0.00,0.09,0.15,0.09,0.17,0.11,0.20,0.13,0.20,0.20,0.00,0.20,0.00,

0.15,0.00,0.15,0.25,0.15,0.25,0.15,0.30,0.20,0.30,0.30,0.00,0.30,0.00,

0.15,0.00,0.15,0.25,0.15,0.25,0.15,0.30,0.20,0.30,0.30,0.00,0.30,0.00};

    double bamort[6][21][8]={
/*BO*/  .9,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        .9,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        .8,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        .7, .9,1.0,1.0,1.0,1.0,1.0,1.0,
        .5, .8,1.0,1.0,1.0,1.0,1.0,1.0,
          .4, .7, .9,1.0,1.0,1.0,1.0,1.0,
          .4, .7, .8,1.0,1.0,1.0,1.0,1.0,
          .3, .6, .8,1.0,1.0,1.0,1.0,1.0,
          .3, .6, .8,1.0,1.0,1.0,1.0,1.0,
          .2, .5, .7, .9,1.0,1.0,1.0,1.0,
          .2, .5, .7, .9,1.0,1.0,1.0,1.0,
          .2, .4, .6, .9,1.0,1.0,1.0,1.0,
          .2, .4, .6, .8,1.0,1.0,1.0,1.0,
          .2, .3, .5, .8,1.0,1.0,1.0,1.0,
          .1, .3, .5, .8, .9,1.0,1.0,1.0,
          .1, .3, .5, .8, .9,1.0,1.0,1.0,
          .1, .3, .5, .7, .9,1.0,1.0,1.0,
          .1, .2, .4, .7, .8,1.0,1.0,1.0,
          .1, .2, .4, .6, .8,1.0,1.0,1.0,
          .1, .2, .4, .6, .8, .9,1.0,1.0,

/*DF*/   1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
         1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
          .6,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
          .5, .9,1.0,1.0,1.0,1.0,1.0,1.0,
          .3, .6,1.0,1.0,1.0,1.0,1.0,1.0,
          .3, .3, .9,1.0,1.0,1.0,1.0,1.0,
          .2, .2, .8,1.0,1.0,1.0,1.0,1.0,
          .1, .1, .6,1.0,1.0,1.0,1.0,1.0,
          .1, .1, .4, .9,1.0,1.0,1.0,1.0,
          .1, .1, .2, .9, .9, .9, .9, .9,
          .1, .1, .2, .7, .9, .9, .9, .9,
          .1, .1, .1, .6, .9, .9, .9, .9,
          .0, .0, .1, .5, .9, .9, .9, .9,
          .0, .0, .1, .4, .8, .9, .9, .9,
          .0, .0, .0, .3, .8, .9, .9, .9,
          .0, .0, .0, .2, .7, .9, .9, .9,
          .0, .0, .0, .1, .6, .9, .9, .9,
          .0, .0, .0, .1, .6, .8, .8, .8,
          .0, .0, .0, .1, .5, .8, .8, .8,
          .0, .0, .0, .1, .4, .8, .8, .8,
          .0, .0, .0, .1, .3, .7, .8, .8,

/*HW*/ 1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
         1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
          .8,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
          .8,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
          .8, .9,1.0,1.0,1.0,1.0,1.0,1.0,
          .7, .9,1.0,1.0,1.0,1.0,1.0,1.0,
          .7, .8,1.0,1.0, .9,1.0,1.0,1.0,
          .6, .8,1.0,1.0, .9,1.0,1.0,1.0,
          .6, .7, .9,1.0, .9,1.0,1.0,1.0,
          .6, .7, .9,1.0, .8,1.0,1.0,1.0,
          .5, .7, .9,1.0, .8,1.0,1.0,1.0,
          .5, .7, .9,1.0, .8,1.0,1.0,1.0,
        .5, .7, .9,1.0, .8,1.0,1.0,1.0,
        .4, .7, .9,1.0, .8,1.0,1.0,1.0,
        .4, .7, .9,1.0, .8,1.0,1.0,1.0,
        .3, .4, .6,1.0, .5,1.0,1.0,1.0,
        .3, .4, .6, .9, .5,1.0,1.0,1.0,
        .3, .4, .6, .9, .5,1.0,1.0,1.0,
        .3, .3, .5, .8, .4,1.0,1.0,1.0,
        .3, .3, .5, .7, .4,1.0,1.0,1.0,

/*PP*/  1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        .7,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        .6, .6,1.0,1.0,1.0,1.0,1.0,1.0,
        .4, .4,1.0,1.0,1.0,1.0,1.0,1.0,
        .3, .3, .7,1.0,1.0,1.0,1.0,1.0,
        .2, .2, .3,1.0,1.0,1.0,1.0,1.0,
        .2, .2, .2,1.0,1.0,1.0,1.0,1.0,
        .1, .1, .1, .8,1.0,1.0,1.0,1.0,
        .1, .1, .1, .5,1.0,1.0,1.0,1.0,
        .1, .1, .1, .3,1.0,1.0,1.0,1.0,
        .1, .1, .1, .1, .9, .9, .9, .9,
        .1, .1, .1, .1, .9, .9, .9, .9,
        .1, .1, .1, .1, .7, .9, .9, .9,
        .0, .0, .0, .0, .5, .9, .9, .9,
        .0, .0, .0, .0, .4, .9, .9, .9,
        .0, .0, .0, .0, .2, .9, .9, .9,
        .0, .0, .0, .0, .1, .8, .9, .9,
        .0, .0, .0, .0, .1, .8, .9, .9,
        .0, .0, .0, .0, .1, .7, .8, .8,
        .0, .0, .0, .0, .0, .6, .8, .8,

/*SP*/  1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        .7,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        .7, .7,1.0,1.0,1.0,1.0,1.0,1.0,
        .6, .6, .9,1.0,1.0,1.0,1.0,1.0,
        .6, .6, .7,1.0,1.0,1.0,1.0,1.0,
        .5, .5, .5,1.0,1.0,1.0,1.0,1.0,
        .5, .5, .5, .9,1.0,1.0,1.0,1.0,
        .4, .4, .4, .7,1.0,1.0,1.0,1.0,
        .4, .4, .4, .5,1.0,1.0,1.0,1.0,
        .3, .3, .3, .3, .9,1.0,1.0,1.0,
        .3, .3, .3, .3, .8,1.0,1.0,1.0,
        .3, .3, .3, .3, .6,1.0,1.0,1.0,
        .2, .2, .2, .2, .4, .9,1.0,1.0,
        .2, .2, .2, .2, .3, .9,1.0,1.0,
        .2, .2, .2, .2, .2, .8,1.0,1.0,
        .2, .2, .2, .2, .2, .7,1.0,1.0,
        .1, .1, .1, .1, .1, .5, .9,1.0,
        .1, .1, .1, .1, .1, .4, .9,1.0,
        .1, .1, .1, .1, .1, .3, .8,1.0,
            .1, .1, .1, .1, .1, .2, .8,1.0,

/*WF*/  1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
         .7,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
         .6, .9,1.0,1.0,1.0,1.0,1.0,1.0,
         .5, .6,1.0,1.0,1.0,1.0,1.0,1.0,
         .4, .4, .9,1.0,1.0,1.0,1.0,1.0,
         .4, .4, .7,1.0,1.0,1.0,1.0,1.0,
         .3, .3, .5, .9,1.0,1.0,1.0,1.0,
         .2, .2, .3, .8,1.0,1.0,1.0,1.0,
         .2, .2, .2, .7,1.0,1.0,1.0,1.0,
         .2, .2, .2, .5, .9,1.0,1.0,1.0,
         .1, .1, .1, .4, .8,1.0,1.0,1.0,
         .1, .1, .1, .3, .7, .9,1.0,1.0,
         .1, .1, .1, .2, .6, .9,1.0,1.0,
         .1, .1, .1, .1, .5, .9, .9, .9,
         .1, .1, .1, .1, .4, .8, .9, .9,
         .1, .1, .1, .1, .3, .7, .9, .9,
         .1, .1, .1, .1, .2, .6, .9, .9,
         .0, .0, .0, .1, .2, .5, .8, .9,
         .0, .0, .0, .0, .1, .4, .8, .9,
         .0, .0, .0, .0, .1, .3, .7, .9};

    if (pf==2)
    {
        pf=0;
        signaltoprint=1;
    }

//  for (sdctr=0; sdctr<goalcombos; sdctr++)
//      gc[sdctr]=1;

//  gc[gcctr]=1.5;

// insects
    pba=0;
    for (sdctr=0; sdctr<lines; sdctr++)
    {
        pba+=ba[sdctr]*restpa[sdctr];
        if (sp[sdctr]==1)
            dfpba+=ba[sdctr]*restpa[sdctr];
        else if (sp[sdctr]==9)
            wfpba+=ba[sdctr]*restpa[sdctr];
        else if (sp[sdctr]==3 || sp[sdctr]==5 || sp[sdctr]==7)
            ppba+=ba[sdctr]*restpa[sdctr];

    }

    if (pba>wfbat[pag])
    {
        wfbadev=pba-wfbat[pag];
        if (wfbadev>wfpba)
            wfbadev=wfpba;
    }
    if ((pba-wfbadev)>dfbat[pag])
    {
        dfbadev=pba-wfbadev-dfbat[pag];
        if (dfbadev>dfpba)
            dfbadev=dfpba;
    }
    if ((pba-wfbadev-dfbadev)>pbat[pag])
    {
        pbadev=pba-wfbadev-dfbadev-pbat[pag];
        if (pbadev>ppba)
            pbadev=ppba;
    }

    if (signaltoprint)
    {
        fprintf(editfp,"%4.2f %4.2f %4.2f ",pba,dfpba,ppba);
        tinsectdev+=dfpba+ppba+wfpba;
    }

    wfbadev=pow(wfbadev,4);        // just squared each of the basal area deviations, not
the total
    dfbadev=pow(dfbadev,4);
    pbadev=pow(pbadev,4);

    if (pba>0)
        insectdev=(wfbadev+dfbadev+pbadev)/pba*100;

// wildlife
    verticalvar=verticalcomplexity(ht,cr,restpa);


    if (gcctr==4)
    {
        if (verticalvar<verticalvargoal)
            wildlifedev=fabs((verticalvargoal-verticalvar)/verticalvargoal*100);
    }
    else
    {
        if (verticalvar>verticalvargoal)
            wildlifedev=fabs((verticalvargoal-verticalvar)/188/*verticalvargoal*/*100);
    }

    if (signaltoprint)
    {
        fprintf(editfp,"%4.2f ",verticalvar);
        tverticaldev+=wildlifedev;
    }
    wildlifedev=pow(wildlifedev,2);

    if (pf==0)
    {
        snags=0;
        for (a=2; a<5; a++)
            for (b=0; b<4; b++)
                snags+=reportsnag[a][b];

        if (per==0)
        {
            for (sdctr=0; sdctr<lines; sdctr++)
            {
                if (sdensity[sdctr]>0.14 && dbh[sdctr]>15)
                    snags+=morttpa[sdctr];
            }
        }

        if (snags<snagsgoal)
            snagsdev=(snagsgoal-snags)*20;
        snagsdev=pow(snagsdev,2);

        if (signaltoprint)
        {
            fprintf(editfp,"%4.2f ",snags);
            tsnagsdev+=snagsdev;
        }

        dwdlinear=0;
        for (a=2; a<5; a++)
            for (b=0; b<4; b++)
                for (sdctr=0; sdctr<1; sdctr++)
                    dwdlinear+=reportdwd[sdctr][a][b];

        if (per==0)
        {
            for (sdctr=0; sdctr<dwdlines; sdctr++)
            {
                if (dwddensity[sdctr][0][0]>0.14 && dwdld[sdctr][0][0][0]>11 &&
                    dwdln[sdctr][0][0][0]>6)
                    dwdlinear+=dwdln[sdctr][0][0][0]*dwdpa[sdctr][0][0][0];
            }
        }


        if (dwdlinear<dwdgoal)
            dwddev=(dwdgoal-dwdlinear)/10;
        dwddev=pow(dwddev,2);
```

```
            if (signaltoprint)
            {
                fprintf(editfp,"%4.2f ",dwdlinear);
                tdwddev+=dwddev;
            }


        }


    // fire
        for (sdctr=0; sdctr<lines; sdctr++)
        {
            diam=(int)((dbh[sdctr]+0.5)/2);
            if (diam>20)
                diam=20;

            bafl=int((flameln+0.5)/2);
            if (bafl>7)
                bafl=7;

            if (sp[sdctr]==0)
                basp=0;
            else if(sp[sdctr]==1 || sp[sdctr]==2)
                basp=1;
            else if(sp[sdctr]==3|| sp[sdctr]==5)
                basp=3;
            else if(sp[sdctr]==4|| sp[sdctr]==8)
                basp=2;
            else if(sp[sdctr]==6|| sp[sdctr]==9)
                basp=5;
            else
                basp=4;

            bamortpct+=bamort[basp][diam][bafl]*restpa[sdctr]*ba[sdctr];
            totaltrees+=restpa[sdctr]*ba[sdctr];
        }

        if (totaltrees>0)
            bamortpct=bamortpct/totaltrees;
        else
            bamortpct=0;

        vegclass=vegclassification(dbh,restpa,cw,sp,ba);
        covertype=(int)vegclass/100;
        qmdint=(int)(vegclass-covertype*100)/10;
        coverint=(int)(vegclass-covertype*100-qmdint*10);
        covertype-=1;
    torching=standhtlcfn(ht,restpa,cr,flameln,pf,signaltoprint);

    fbi=50/15*flameln +30*torching +20*fabs((cbd[covertype][qmdint][coverint]-
0.1)/0.2);
    fei=bamortpct*100;

    if (signaltoprint)
    {
        fprintf(editfp,"%4.2f %4.2f ",fbi,fei);
    //  fprintf(editfp,"\n%4.2f %4.2f
%4.2f\n",flameln,torching,cbd[covertype][qmdint][coverint]);
    }

    if (signaltoprint)
    {
        tfei+=fei;
        tfbi+=fbi;
    }

    if (fbi<=fbigoal)
        fbi=0;
    if (fei<=fbigoal)
        fei=0;
    fbi=pow(fbi,2);
    fei=pow(fei,2);


// fish
    fishtrees=0;
    snagtrees=0;
    for (sdctr=0; sdctr<lines; sdctr++)
    {
        if (dbh[sdctr]>16 && dbh[sdctr]<30)
            snagtrees+=restpa[sdctr];
        if (dbh[sdctr]>=30)
            fishtrees+=restpa[sdctr];
    }

    if (snagtrees<snagsdev)
        snagtreesdev=snagsdev-snagtrees;

    if (fishtrees<=fishtreesgoal)
        fishtreesdev=(fishtreesgoal-fishtrees)*20;
    if (signaltoprint)
    {
        fprintf(editfp,"%4.2f ",fishtrees);
        tfishtreesdev+=fishtreesdev;
    }
    fishtreesdev=pow(fishtreesdev,2);// try to make fishtrees more important <> raised to
the 4
```

```
                                        // doesn't work

    if (rma)
    {   if (cover<ccgoal)
            ccdev=ccgoal-cover;
        ccdev=pow(ccdev,2);
    }

//  if (objective<10)
//      pnvdev=pow(10-objective,4);

// final
    sumdev=gc[3]*insectdev +gc[4]*fishtreesdev +gc[4]*ccdev
+ksnag*gc[1]*gc[2]*snagsdev
        +gc[0]*fbi +gc[0]*fei +gc[1]*gc[2]*wildlifedev
        +kdwd*gc[1]*gc[2]*dwddev +kstree*gc[1]*gc[2]*snagtreesdev;

    if (signaltoprint)
    {
/*      fprintf(editfp,"\nIn per %d, mode %d\n",per,pf);
        fprintf(editfp,"dfbadev %4.2f wfbadev %4.2f pbadev %4.2f insectdev %4.2f \n",
            dfbadev, wfbadev, pbadev, insectdev);
        fprintf(editfp,"fishtrees %4.2f ccdev %4.2f\n",fishtreesdev,ccdev);
        fprintf(editfp,"vertcomplex %4.2f snagsdev %4.2f dwddev
%4.2f\n",wildlifedev,snagsdev,dwddev);
        fprintf(editfp,"bamortpct %4.2f cbd %4.2f
",bamortpct,cbd[covertype][qmdint][coverint]);
        fprintf(editfp,"torching %d firedev %4.2f\n",torching,firedev);
        fprintf(editfp,"sumdev %4.2f\n\n",sumdev);
*/
        fprintf(editfp,"%4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f
%d",fishtreesdev,
            ccdev,wildlifedev,
            snagsdev,dwddev, fbi, fei, insectdev,per+1);

        fprintf(editfp," fl %4.2f torch %4.2f cbd %4.2f
",flameln,torching,cbd[covertype][qmdint][coverint]);
        fprintf(editfp,"\n");
    }

//  gc[gcctr]=1;

    return sumdev;
}

VEGCLASS
#include "var.h"
extern int lines;
double cover;
```

```
// make sure the report species is passed

int vegclassification(double dbh[], double tpa[], double cw[], int sp[], double ba[])
{
    double standba=0, adjstandba=0, standtpa=0, adjstandtpa=0, qmd, adjqmd;
    double spcover[totsp];
    int vctr, spcovertype, coverint, qmdint, vegclass;

    for (vctr=0; vctr<totsp; vctr++)
        spcover[vctr]=0;

    for (vctr=0; vctr<lines; vctr++)
    {
        if (dbh[vctr]>=1);
        {
            standba+=ba[vctr]*tpa[vctr];
            standtpa+=tpa[vctr];
        }
    }
    qmd=pow((standba/(0.005454154*standtpa)),0.5);
    cover=0;

    for (vctr=0; vctr<lines; vctr++)
    {
        if (dbh[vctr]>=qmd)
        {
            adjstandba+=ba[vctr]*tpa[vctr];
            adjstandtpa+=tpa[vctr];
            cover+=pow((cw[vctr]/2),2)*pi*tpa[vctr];
            spcover[sp[vctr]]+=pow((cw[vctr]/2),2)*pi*tpa[vctr];
        }
    }
    adjqmd=pow((adjstandba/(0.005454154*adjstandtpa)),0.5);

    for (vctr=0; vctr<totsp; vctr++)
        spcover[vctr]=spcover[vctr]/cover*100;//435.6;
    cover=cover/435.6;

    if ((spcover[0] +spcover[1] +spcover[2] +spcover[3] +spcover[4] +spcover[5]
+spcover[6]
         +spcover[7] +spcover[8] +spcover[9])<20)
         spcovertype=6;                                        //open
    else if ((spcover[0] +spcover[4] +spcover[8])>30)
    {
        if ((spcover[7] +spcover[1] +spcover[2] +spcover[3] +spcover[9] +spcover[5]
+spcover[6])
            >30)
            spcovertype=1;                                     // CH
        else if ((spcover[4] +spcover[8])>50)
            spcovertype=3;                                     // EH
            else
                spcovertype=2;                          // DH
        }
        else if ((spcover[6] +spcover[9])>50)
        {
            if (spcover[6]>spcover[9])
                spcovertype=8;                          // RF
            else
                spcovertype=9;                          // WF
        }
        else if ((spcover[3] +spcover[5] +spcover[7])<50)
            spcovertype=5;                              // MC
        else if ((spcover[5] +spcover[7])>spcover[3])
            spcovertype=7;                              // pine
        else
            spcovertype=4;                              // KP

        if (cover<=60)
            coverint=0;
        else
            coverint=1;

        if (adjqmd<5)
            qmdint=0;
        else if (adjqmd<9)
            qmdint=1;
        else if (adjqmd<15)
            qmdint=2;
        else if (adjqmd<21)
            qmdint=3;
        else if (adjqmd<25)
            qmdint=4;
        else if (adjqmd<32)
            qmdint=5;
        else
            qmdint=6;

        if (qmdint==0 || qmdint==6 || qmdint==5)
            coverint=0;

        vegclass=spcovertype*100 +qmdint*10 + coverint;

        return (vegclass);
}

VOLHRV
#include "var.h"
extern int lines;
extern int sp[linesor];
extern double vol[linesor];
extern double volabhco1[totsp], volabhco2[totsp], dib1co1[totsp], dib1co2[totsp],
dib1co3[totsp];
extern double dibco1[totsp], dibco2[totsp];

void volharvest(double tpa[],double dbh[], double ht[])
{
    int vhctr;
    double htabh,volabh,volbbh,dib, dib1, r;
    double k1,k2,k3=0.000162,k4,k5,k6,k7;

    for (vhctr=0; vhctr<lines; vhctr++)

    {   htabh=ht[vhctr]-4.5;



        if (vhctr==40)
            volbbh=0;

        if (ht[vhctr]>4.5)
        //
    volabh=volabhco1*(pow(htabh/dbh[vhctr],volabhco2))*(pow(dbh[vhctr],2))*htabh;
            volabh=volabhco1[sp[vhctr]]*(pow(htabh/dbh[vhctr],volabhco2[sp[vhctr]]))
            *(pow(dbh[vhctr],2))*htabh;
        else
            volabh=0;



    //  dib=dibco1*(pow(dbh[vhctr],dibco2));
    //  dib1=dib1co1+dib1co2*(pow(dbh[vhctr],dib1co3));
        dib=dibco1[sp[vhctr]]*(pow(dbh[vhctr],dibco2[sp[vhctr]]));

    dib1=dib1co1[sp[vhctr]]+dib1co2[sp[vhctr]]*(pow(dbh[vhctr],dib1co3[sp[vhctr]]));
        if (dib1>0)
        {
            r=pow((dib/dib1),2/3);

            k1=0.25*3.14156*pow(dib1,2);
            k2=(1/43904)*(729+81*r +297*pow(r,2) +265*pow(r,3));
        //  k3=0.;
            k4=pow((4.5-r),3);
            k5=1.5*pow((4.5-r),2)*(1-r);
            k6=(4.5-r)*pow((1-r),2);
            k7=pow((1-r),3);

            volbbh=k1*(k2-k3*(k4-k5+k6-k7));
            if (volbbh<0)
                volbbh=0;
                vol[vhctr]=volbbh+volabh;
            }
            else
                vol[vhctr]=0;


            //fprintf(growfp,"\nvol=%f",vol[vhctr]);

        }

    }
```

VOLS

```
#include "var.h"
extern int lines;
extern double bfmind[totsp];
extern double bftopd[totsp];
extern double bfv[linesor], bfrevenue, bfrev[linesor];

void vols(double volstpa[], double volsdbh[], double volsdg[], int sp[], double volsht[])
{
    int vctr;
    double brat, d;

    for (vctr=0; vctr<lines; vctr++)
    {
        bfv[vctr]=0;
        bfrev[vctr]=0;

        if (volstpa[vctr]>0)
        {
            brat=bratio(sp[vctr],volsdbh[vctr]);
        //  d=volsdbh[vctr]+volsdg[vctr]/brat;
            d=volsdbh[vctr];

            if (d>=bfmind[sp[vctr]] && d>bftopd[sp[vctr]])
            {
                bfv[vctr]=bfvol(sp[vctr],d,volsht[vctr],brat);
                bfrev[vctr]=bfrevenue;
            }
        }
    }
}
```

## Appendix B: Goal Measures for All Goal Emphases

| Per | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 47.2 | 7.99 | 7.99 | 7.99 | 40.18 | 7.99 | 7.99 | 7.99 | 7.99 |
| 2 | 44.2 | 3.99 | 7.99 | 11.98 | 7.99 | 7.99 | 3.99 | 4.96 | 4.98 |
| 3 | 11.2 | 5.97 | 9.97 | 14.95 | 9.97 | 10.96 | 6.96 | 7.93 | 7.95 |
| 4 | 11.2 | 2.99 | 11.95 | 37.93 | 11.95 | 5.94 | 2.99 | 3.91 | 9.92 |
| 5 | 5.2 | 9.97 | 16.93 | 39.91 | 7.99 | 38.91 | 4.96 | 6.88 | 9.89 |
| 6 | 6.2 | 5.99 | 18.91 | 41.89 | 9.97 | 56.88 | 31.99 | 37.86 | 39.86 |
| 7 | 4.2 | 5.96 | 50.89 | 43.87 | 5.95 | 48.86 | 0.99 | 41.82 | 10.99 |
| 8 | 6.2 | 32.99 | 57.87 | 45.85 | 21.93 | 50.83 | 34.97 | 41.79 | 2.99 |
| 9 | 6.2 | 34.97 | 59.85 | 51.83 | 23.91 | 52.81 | 2.99 | 44.76 | 2.99 |
| 10 | 34.2 | 32.99 | 61.83 | 53.81 | 55.89 | 49.79 | 32.99 | 48.74 | 33.99 |
| 11 | 1.2 | 4.96 | 63.81 | 53.79 | 57.87 | 51.76 | 0.99 | 51.71 | 0.99 |
| 12 | 32.19 | 2.99 | 53.79 | 52.77 | 44.85 | 59.73 | 30.99 | 46.68 | 30.99 |
| 13 | 15.99 | 3.97 | 51.76 | 59.75 | 61.83 | 61.71 | 3.98 | 16.65 | 15.99 |
| 14 | 40.99 | 1.99 | 64.73 | 58.73 | 60.8 | 50.69 | 2.99 | 32.99 | 2.99 |
| 15 | 15.99 | 3.97 | 56.71 | 61.71 | 67.77 | 53.66 | 4.96 | 35.96 | 2.99 |
| 16 | 0.99 | 31.99 | 52.69 | 62.69 | 63.75 | 65.63 | 31.99 | 38.93 | 32.99 |
| 17 | 1.98 | 34.96 | 51.67 | 64.67 | 60.72 | 68.6 | 1.99 | 2.99 | 0.99 |
| 18 | 31.98 | 31.99 | 44.65 | 69.65 | 57.69 | 74.58 | 31.99 | 32.98 | 30.99 |
| 19 | 2.98 | 4.96 | 43.63 | 68.63 | 24.66 | 64.55 | 5.99 | 3.99 | 15.99 |
| 20 | 31.98 | 1.99 | 48.6 | 70.61 | 21.63 | 75.52 | 31.99 | 35.96 | 3.99 |

Table 1. FBI by Period for All Goal Emphases.

| Per | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 16.85 | 16.85 | 16.85 | 16.85 | 16.85 | 16.85 | 16.85 | 16.85 | 16.85 |
| 2 | 14.09 | 18.36 | 17.47 | 14.86 | 16.74 | 18.38 | 18.36 | 11.92 | 18.52 |
| 3 | 13.59 | 11.73 | 16.32 | 19.43 | 15.98 | 18.67 | 16.88 | 13.95 | 25.71 |
| 4 | 16.54 | 13.73 | 21.6 | 21.42 | 21.71 | 40.95 | 16.52 | 14.01 | 27.74 |
| 5 | 15.02 | 9.51 | 22.93 | 21.61 | 17.66 | 34.37 | 20.25 | 17.15 | 31.65 |
| 6 | 22.36 | 30.24 | 22.76 | 28.14 | 17.02 | 48.4 | 24.38 | 24.82 | 36.99 |
| 7 | 28.5 | 24.16 | 34.99 | 32.38 | 21.79 | 46.64 | 20.84 | 33.82 | 22.7 |
| 8 | 17.83 | 33.11 | 37.75 | 35.13 | 23.09 | 43.45 | 29.27 | 39.93 | 11.98 |
| 9 | 21.38 | 21.14 | 44.29 | 63.04 | 23.64 | 44.54 | 29.44 | 41.89 | 14.58 |
| 10 | 30.45 | 28.5 | 78.82 | 64.72 | 48.61 | 45.31 | 32.19 | 60.91 | 17.91 |
| 11 | 23.95 | 20 | 82.89 | 62.6 | 48.09 | 64.85 | 29.47 | 68.53 | 20.35 |
| 12 | 27.83 | 24.71 | 74.94 | 87.65 | 51.07 | 60.51 | 29.93 | 25.23 | 24.36 |
| 13 | 31.64 | 14.05 | 81.33 | 82.29 | 86.68 | 99.5 | 25.98 | 25.19 | 15.21 |
| 14 | 26.69 | 28.46 | 89.66 | 82.81 | 84.72 | 99.29 | 26.32 | 21.51 | 16.47 |
| 15 | 22.87 | 13.1 | 81.03 | 95.8 | 99.44 | 98.92 | 10.84 | 21.68 | 18.04 |
| 16 | 25.84 | 44.21 | 71.08 | 94.96 | 81.87 | 100 | 18.97 | 25.53 | 19.82 |
| 17 | 21.83 | 21.15 | 72.86 | 96.73 | 43.43 | 100 | 24.24 | 21.36 | 20.23 |
| 18 | 27.86 | 31.84 | 64.34 | 97.63 | 38.1 | 100 | 30.1 | 27.9 | 24.12 |
| 19 | 23.42 | 20.88 | 62.84 | 83.21 | 20.91 | 100 | 24.38 | 38.67 | 40.36 |
| 20 | 16.89 | 27.2 | 57.79 | 97.34 | 20.98 | 100 | 32.73 | 37.62 | 39.63 |

Table 2. FEI by Period for All Goal Emphases.

| Per | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 133.3 | 133.3 | 133.3 | 133.3 | 133.3 | 133.3 | 133.3 | 133.3 | 133.3 |
| 2 | 77.94 | 13.3 | 113.12 | 47.18 | 148.47 | 161.08 | 13.3 | 49.58 | 14.31 |
| 3 | 65.92 | 6.54 | 97.75 | 48.36 | 96.35 | 166.21 | 13.43 | 50.48 | 18.55 |
| 4 | 71.01 | 9.85 | 104.61 | 45.4 | 107.68 | 178.33 | 15.06 | 34.13 | 19.71 |
| 5 | 65.08 | 11.09 | 93.56 | 47.45 | 98.76 | 202.17 | 88.24 | 38.69 | 87.02 |
| 6 | 42.61 | 9.51 | 108.87 | 47.4 | 110.61 | 182.02 | 115.18 | 32.2 | 119.15 |
| 7 | 47.11 | 18.74 | 108.31 | 57.32 | 104.4 | 185.64 | 183.76 | 39.47 | 184.83 |
| 8 | 49.29 | 19.17 | 103.99 | 44.3 | 115.4 | 172.81 | 21.48 | 46.68 | 8.11 |
| 9 | 29.47 | 18 | 103.49 | 47.18 | 108.74 | 183.33 | 38.25 | 73.4 | 30.01 |
| 10 | 20.44 | 23.27 | 107.74 | 53.24 | 119.78 | 166.3 | 44.48 | 63.59 | 40.9 |
| 11 | 22.18 | 20.43 | 109.62 | 48.52 | 110.01 | 98.83 | 149.44 | 81.37 | 105.92 |
| 12 | 23.56 | 23.39 | 94.91 | 57.26 | 120.44 | 102.85 | 185.38 | 86.95 | 129.52 |
| 13 | 175.17 | 16.42 | 85.99 | 45.47 | 120.05 | 58.07 | 25.63 | 125.24 | 19.16 |
| 14 | 158.12 | 15.77 | 89.36 | 55.18 | 124.62 | 52.23 | 26.91 | 50 | 20.11 |
| 15 | 249.15 | 18.72 | 27.8 | 51.99 | 116.87 | 48.93 | 12.39 | 61.91 | 26.13 |
| 16 | 321.32 | 0 | 32.37 | 43.9 | 114.47 | 51.89 | 10.69 | 69.59 | 31.15 |
| 17 | 321.76 | 4.07 | 31.68 | 48.32 | 113.77 | 60.8 | 108.71 | 90.06 | 102.3 |
| 18 | 299.74 | 18.78 | 30.76 | 58.5 | 109.53 | 69.26 | 128.49 | 50.69 | 128.9 |
| 19 | 293.57 | 22.26 | 30.29 | 52.61 | 107.84 | 88.32 | 252.02 | 58.02 | 16.01 |
| 20 | 296.49 | 24.92 | 18.34 | 59.02 | 105.16 | 106.3 | 12.27 | 65.13 | 16.76 |

Table 3. Vertical Complexity by Period for All Goal Emphases.

| Per | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0.04 | 0.01 | 0.04 | 0.03 | 0.05 | 0.05 | 0.01 | 0.03 | 0.01 |
| 3 | 0.1 | 0.01 | 0.09 | 5.33 | 5.35 | 0.1 | 0.01 | 5.33 | 5.3 |
| 4 | 0.16 | 0.02 | 0.14 | 5.37 | 5.37 | 0.12 | 0.02 | 5.34 | 5.31 |
| 5 | 0.19 | 0.02 | 0.17 | 5.41 | 5.38 | 0.14 | 0.03 | 5.35 | 5.31 |
| 6 | 0.19 | 0.02 | 0.19 | 5.12 | 5.06 | 0.17 | 0.04 | 5.02 | 4.98 |
| 7 | 0.18 | 0.02 | 0.2 | 4.64 | 4.57 | 0.16 | 0.05 | 4.51 | 4.79 |
| 8 | 0.18 | 0.02 | 0.2 | 4.87 | 4.86 | 0.15 | 0.05 | 4.84 | 4.84 |
| 9 | 0.16 | 0.02 | 0.21 | 4.89 | 4.9 | 0.14 | 0.04 | 4.89 | 4.89 |
| 10 | 0.14 | 0.01 | 0.2 | 4.84 | 4.85 | 0.11 | 0.04 | 4.84 | 4.83 |
| 11 | 0.09 | 0.01 | 0.19 | 2.14 | 2.16 | 0.09 | 0.03 | 1.69 | 2.14 |
| 12 | 0.05 | 0 | 0.18 | 5.2 | 5.21 | 0.07 | 0.02 | 1.63 | 5.24 |
| 13 | 0.03 | 0 | 0.17 | 5.04 | 5.12 | 0.05 | 0.02 | 1.55 | 5.16 |
| 14 | 0.02 | 0 | 0.18 | 4.86 | 4.99 | 0.03 | 0.01 | 1.47 | 5.04 |
| 15 | 0.02 | 0 | 0.18 | 4.91 | 4.84 | 0.03 | 0 | 1.38 | 4.58 |
| 16 | 0 | 0 | 0.16 | 4.47 | 4.18 | 0.02 | 0 | 0.95 | 4.04 |
| 17 | 0 | 0 | 0.15 | 4.68 | 3.74 | 0 | 0 | 0.91 | 3.72 |
| 18 | 0 | 0 | 0.14 | 4.74 | 3.56 | 0 | 0 | 0.88 | 3.43 |
| 19 | 0 | 0 | 0.13 | 4.7 | 3.4 | 0 | 0 | 0.84 | 2.88 |
| 20 | 0 | 0 | 0.12 | 3.32 | 5.2 | 0 | 0 | 5.27 | 1.31 |

Table 4. Snags by Period for All Goal Emphases.

| Per | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0.11 | 0.05 | 0.12 | 0.08 | 0.12 | 0.12 | 0.05 | 0.08 | 0.02 |
| 6 | 0.51 | 0.12 | 0.35 | 4.05 | 4.12 | 0.38 | 0.12 | 4.05 | 3.9 |
| 7 | 1.25 | 0.19 | 0.73 | 5.57 | 5.63 | 0.78 | 0.2 | 5.49 | 5.21 |
| 8 | 1.59 | 0.22 | 0.93 | 4.56 | 4.4 | 1.04 | 0.25 | 4.22 | 3.91 |
| 9 | 1.51 | 0.23 | 1.05 | 4.04 | 3.56 | 1.34 | 0.33 | 3.31 | 2.96 |
| 10 | 1.7 | 0.21 | 1.4 | 4.33 | 3.63 | 1.51 | 0.47 | 3.12 | 3.55 |
| 11 | 1.41 | 0.16 | 1.4 | 3.89 | 3.53 | 1.4 | 0.55 | 2.36 | 2.17 |
| 12 | 0.85 | 0.12 | 1.38 | 6.57 | 6.79 | 1.33 | 0.56 | 3.34 | 2.78 |
| 13 | 0.63 | 0.07 | 1.51 | 8.38 | 8.97 | 1.06 | 0.45 | 4.47 | 4.12 |
| 14 | 0.47 | 0.03 | 1.64 | 10.01 | 11.54 | 0.89 | 0.31 | 6.6 | 9.6 |
| 15 | 0.26 | 0.02 | 1.77 | 24.53 | 14.26 | 0.81 | 0.25 | 9.64 | 28.61 |
| 16 | 0.24 | 0 | 2 | 26.34 | 23.58 | 0.7 | 0.13 | 10.02 | 50 |
| 17 | 0.27 | 0 | 2.23 | 19.63 | 31.5 | 0.67 | 0.09 | 10.06 | 62.23 |
| 18 | 0.29 | 0 | 2.47 | 30.48 | 36.77 | 0.68 | 0 | 11.22 | 59.33 |
| 19 | 0.28 | 0 | 2.59 | 25.95 | 41.31 | 0.65 | 0 | 11.38 | 52.53 |
| 20 | 0.29 | 0 | 2.8 | 22.1 | 39.23 | 0.7 | 0 | 11.64 | 48.54 |

Table 5. DWD by Period for All Goal Emphases.

| Per | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 257.7 | 257.7 | 257.7 | 257.7 | 257.7 | 257.7 | 257.7 | 257.7 | 257.7 |
| 2 | 252.45 | 37.64 | 276.7 | 188.6 | 282.96 | 298.21 | 37.64 | 189.87 | 39.38 |
| 3 | 263.87 | 37.83 | 296.63 | 206.43 | 287.55 | 328.53 | 40.92 | 206.79 | 42.11 |
| 4 | 273.42 | 36.36 | 291.76 | 194.52 | 259.91 | 146.42 | 46.63 | 111.47 | 47.13 |
| 5 | 282.1 | 37 | 242.74 | 204.87 | 242.34 | 171.14 | 59.44 | 123.02 | 58.92 |
| 6 | 134.57 | 12.95 | 246.88 | 204.1 | 256.09 | 197.71 | 92.01 | 75.03 | 89.21 |
| 7 | 98.58 | 13.58 | 261.24 | 188.61 | 241.53 | 181.11 | 124.29 | 84.31 | 118.51 |
| 8 | 110.38 | 23.39 | 251.62 | 184.99 | 223.27 | 210.12 | 27.44 | 99.5 | 48.8 |
| 9 | 59.81 | 21.64 | 221.88 | 156.13 | 202.69 | 216.15 | 33.91 | 117.54 | 52.74 |
| 10 | 21.16 | 23.8 | 212.88 | 141.56 | 195.07 | 205.73 | 49.31 | 64.06 | 65.34 |
| 11 | 25.69 | 25.63 | 223.38 | 137.61 | 195.4 | 130.09 | 72.41 | 82.7 | 73.38 |
| 12 | 34.77 | 33.7 | 192.02 | 144.4 | 204.37 | 148.83 | 118.9 | 108.69 | 106.49 |
| 13 | 50.06 | 27.88 | 165.24 | 135.61 | 221.02 | 102.16 | 35.71 | 137.68 | 39.05 |
| 14 | 79.8 | 16.31 | 182.79 | 134.74 | 243.92 | 72.9 | 40.21 | 90.19 | 42.88 |
| 15 | 116.16 | 18.19 | 106.88 | 138.8 | 268.04 | 82.37 | 14.11 | 106.29 | 48.45 |
| 16 | 134.53 | 9.85 | 114.27 | 116.08 | 290.74 | 95.41 | 19.4 | 127.84 | 60.42 |
| 17 | 177.88 | 10.13 | 124.47 | 121.43 | 312.83 | 112.08 | 31.07 | 150.32 | 77.86 |
| 18 | 221.69 | 18.58 | 134.35 | 130.09 | 336.89 | 75.11 | 63.12 | 99.79 | 112.46 |
| 19 | 266.95 | 19.05 | 145.83 | 141.75 | 354.76 | 94.34 | 100.79 | 109.97 | 43.66 |
| 20 | 313.29 | 27.04 | 112.98 | 142.63 | 378.83 | 120.81 | 11.92 | 130.13 | 47.65 |

Table 6. Stand Basal Area Per Acre by Period for All Goal Emphases.

| Per | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1.63 | 1.63 | 1.63 | 0 | 0 | 1.63 | 0 |
| 7 | 0 | 0 | 2.91 | 1.63 | 1.63 | 0 | 1.24 | 4.21 | 2.62 |
| 8 | 0 | 0 | 4.14 | 2.87 | 2.87 | 0 | 0 | 4.21 | 2.61 |
| 9 | 0 | 0 | 4.13 | 2.87 | 2.87 | 0 | 0 | 4.2 | 2.61 |
| 10 | 0 | 0 | 4.11 | 2.87 | 2.87 | 0 | 1.56 | 4.75 | 3.1 |
| 11 | 0 | 0 | 4.1 | 4.43 | 1.24 | 0 | 1.56 | 4.74 | 3.09 |
| 12 | 0 | 0 | 5.64 | 2.8 | 1.24 | 0 | 1.56 | 4.74 | 3.09 |
| 13 | 0 | 0 | 5.63 | 2.8 | 0 | 0 | 0 | 4.73 | 3.09 |
| 14 | 0 | 0 | 5.62 | 2.8 | 0 | 0 | 0 | 4.73 | 3.08 |
| 15 | 0 | 0 | 5.62 | 2.8 | 0 | 0 | 0 | 4.72 | 3.08 |
| 16 | 0 | 0 | 5.61 | 2.8 | 0 | 0 | 0 | 4.71 | 3.08 |
| 17 | 0 | 0 | 5.6 | 1.24 | 0 | 0 | 0 | 4.7 | 3.07 |
| 18 | 0 | 0 | 5.6 | 0 | 0 | 0 | 0 | 4.7 | 3.07 |
| 19 | 0 | 0 | 5.59 | 0 | 0 | 0 | 0 | 4.69 | 3.06 |
| 20 | 0 | 0 | 5.58 | 0 | 0 | 0 | 0 | 4.69 | 3.06 |

Table 7. Large Trees Per Acre by Period for All Goal Emphases.

# Appendix C: Cubic Feet Harvested

| Period | Goal Emphasis | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
| 1 | 71.69 | 3235.95 | 71.69 | 890.75 | | | 3529.64 | 829.29 | 3363.6 |
| 2 | 139.97 | 343.7 | 16.26 | | 120.72 | | | | |
| 3 | 474.81 | | 648.57 | 470.32 | 1220.86 | 4034.29 | | 1939.57 | |
| 4 | 3608.37 | | 1245.89 | 166.24 | 718.8 | | | | |
| 5 | 1088.36 | 613.18 | 444.32 | 450.36 | 247.03 | | | 1510.15 | |
| 6 | 154.95 | | 216.77 | 666.02 | 823.84 | 737.45 | | | |
| 7 | 213.06 | | 715.87 | 182.6 | 773.19 | | 1307.8 | | 561.0 |
| 8 | 522.12 | | 934.86 | 1037.57 | 755.06 | 514.38 | | 961.62 | |
| 9 | 5.53 | 47.6 | 659.15 | 669.13 | 601.86 | 517.48 | | | |
| 10 | 82.63 | 74.08 | 253.72 | 166.58 | 194.16 | 1248.91 | | | |
| 11 | | 115.93 | 669.22 | 203.93 | 298.02 | | | | |
| 12 | | | 742.25 | 405.26 | 131.71 | 1075.29 | 818.45 | | 833.1 |
| 13 | 472.02 | 334.95 | | 410.94 | 55.99 | 557.53 | | 502.86 | |
| 14 | | | 1541.06 | 136.67 | | | | | |
| 15 | 1297.63 | | 17.84 | 467.89 | 24.08 | | | 893.34 | |
| 16 | | | | 207.56 | 21.27 | | | | |
| 17 | 1021.83 | 67.32 | 12.99 | 167.19 | | 786.45 | | | |
| 18 | 621.63 | 65.32 | | 32.2 | 37.6 | | 766.28 | | 796.3 |
| 19 | 521.19 | | 641.68 | 219.12 | | | | | |
| 20 | 9.43 | 28.57 | | 181.56 | 0.04 | | 841.48 | 874.2 | |

Table 8. Cubic Feet Harvested by Period for All Goal Emphases

# Appendix D: Snags Created

| Period | | Goal Emphases | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
| 1 | | | | 5.27 | 5.25 | | | 5.27 | 5.2 |
| 2 | | | | | | | | | |
| 3 | | | | | | | | | |
| 4 | | | | | | | | | |
| 5 | | | | | | | | | 0.3 |
| 6 | | | | 0.66 | 0.73 | | | 0.77 | 0.5 |
| 7 | | | | 0.43 | 0.44 | | | 0.46 | 0.4 |
| 8 | | | | 0.41 | 0.4 | | | 0.42 | 0.4 |
| 9 | | | | 0.46 | 0.45 | | | | 0.4 |
| 10 | | | | 3.16 | 3.14 | | | | 3.1 |
| 11 | | | | | | | | | |
| 12 | | | | | | | | | |
| 13 | | | | 0.44 | | | | | |
| 14 | | | | 0.39 | | | | | |
| 15 | | | | 0.83 | | | | | |
| 16 | | | | 0.62 | | | | | |
| 17 | | | | 0.56 | | | | | |
| 18 | | | | 0.6 | 1.9 | | | | |
| 19 | | | | 1.98 | | | | | |
| 20 | | | | | | | | | |

Table 9. Snags Created by Period for All Goal Emphases

# Appendix E: Vertical Structure

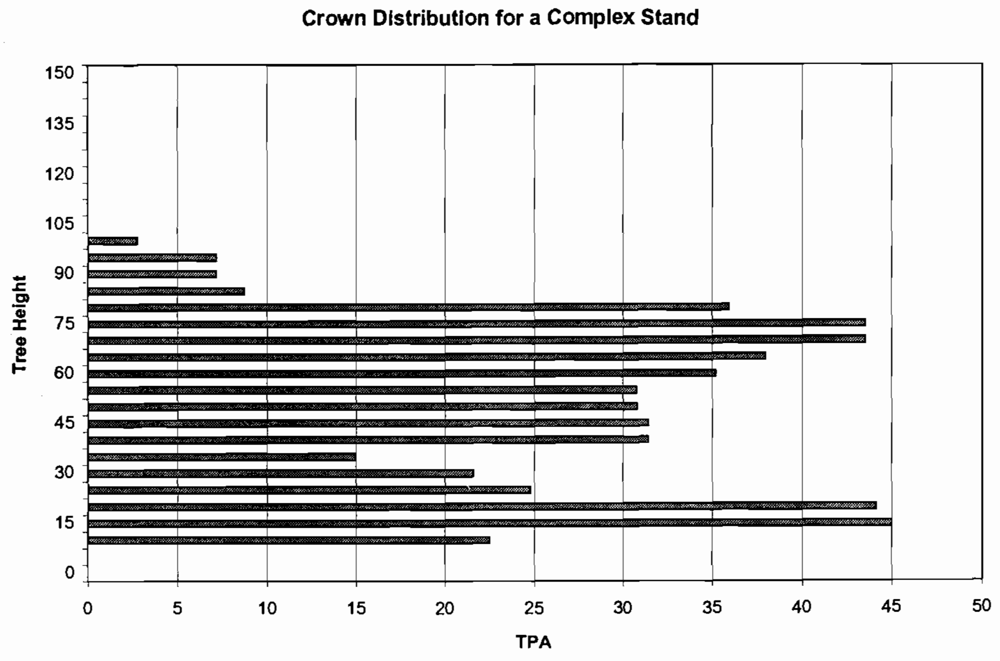


Figure 1. Crown Distribution for a Complex Stand, Standard Deviation = 14

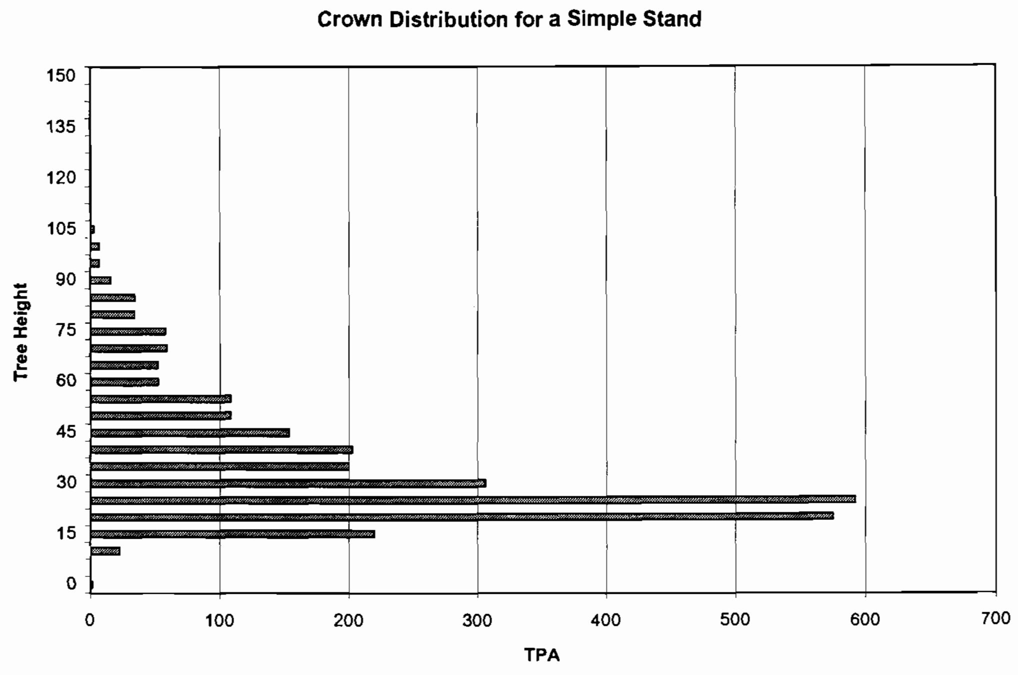


Figure 2. Crown Distribution for a Simple Stand, Standard Deviation = 160