FOREST HARVEST SCHEDULING PROBLEM: STUDYING MATHEMATICAL AND CONSTRAINT PROGRAMMING SOLUTION STRATEGIES

by

Junas Adhikary

B.Sc., Trent University, Peterborough, ON Canada, 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
April 1997

APPROVAL

Name: Junas Adhikary

Degree: Master of Science

Title of thesis: Forest Harvest Scheduling Problem: Studying Mathematical and Constraint Programming Solution Strategies

Examining Committee: Dr. Hassan Aït-Kaci
Chair

Dr. William S. Havens
Senior Supervisor

Dr. Lou Hafer
Supervisor

Dr. Jim Delgrande
External Examiner

Date Approved: January 31, 1997

Abstract

The Forest Harvest Scheduling Problem (FHSP) is an important part of the forest resource management. It is a complex multi-criteria optimization problem. Our review of the optimization techniques applicable to forest harvesting problems in general suggests that long term scheduling is difficult because of the prohibitive size and the complexity inherent to the problem. In this thesis, we study mathematical and constraint programming as both modelling and solving techniques for such problems. We discuss how these solution strategies may accommodate the forest growth and treatment simulation model. The simulation model is an important part of the scheduling system since it makes the solution more realistic and implementable. We give a mixed integer programming (MIP) formulation for the restricted FHSP problem and test it using real-life data from the Norwegian ECOPLAN project. Since larger instances were not solved in a reasonable time, the MIP model alone is not sufficient to solve practical FHSP. We advocate the combined use of the Constraint Satisfaction Problem (CSP) model and the iterative improvement technique as a solution strategy. The simulation model can also be more easily integrated in this strategy than in the MIP model. The iterative improvement techniques will in general benefit from the high quality initial solutions. We show how a CSP formulation can be used to find "good" initial solutions to the problem, based on simulation results from a stand growth and treatment simulator. The initial solution generator can be used as a module in an integrated forest treatment scheduling system which will advance the state-of-the-art in forest harvesting practices.

Acknowledgments

The author is extremely grateful for the continuing support and guidance of his senior supervisor, Dr. William S. Havens. Thanks to his supervisor Dr. Lou Hafer for helping in the mathematical side of the research. I am extremely grateful to Gunnar Misund and Dr. Geir Hasle from SINTEF Oslo for giving me an opportunity to work closely with them on the Norwegian ECOPLAN project and for providing the ECOPLAN prototype data. Thanks to Morten Irgens for initiating this collaboration between ISL and SINTEF. Finally my special thanks goes to all the members of the Intelligent Systems Lab for reading my drafts and helping me improve substantially.

To my parents Gehendra and Sushila Adhikary.

Contents

Approval Page ii
Abstract iii
Acknowledgments iv
Dedication v
List of Tables viii
List of Figures ix
1 Introduction 1
1.1 Problem Definition 3
1.1.1 Growth and Treatment Simulation 7
1.2 Review of Optimized Forest Harvest Scheduling 8
1.2.1 Mathematical Programming 9
1.2.2 Heuristic Optimization 10
1.2.3 Miscellaneous Methodology 13
1.2.4 Summary 13
1.3 Overview of the Thesis 15
2 Formalization of the FHSP 16
2.1 Identifying Cuts 17
2.1.1 0-1 Node-Packing Problem 19
2.2 Consistency Enforcement 22
2.2.1 Row Convexity 24
2.3 Conclusion 25
3 Solving the CCSP using MIP Models 26
3.1 MIP Model I 27
3.2 MIP Model II 28

3.3 Adjacency Constraints Revisited 30
3.4 Experimental Results 31
3.4.1 Initial experiments using *lp_solve* solver 31
3.4.2 Results using *cplex* solver 32
3.5 Integrating LP with Simulation 37
3.6 Conclusion 39
4 Solving the FHSP using a CSP model 40
4.1 Methods Studied for Generating Initial Schedule 41
4.1.1 Linear Programming 41
4.1.2 Mixed Integer Programming 43
4.1.3 The CSP model and Constraint Programming 43
4.1.4 Conclusion 45
4.2 A method based on a CSP model 45
4.2.1 Minconflicts Heuristic 46
4.3 Experimental Results 47
4.3.1 Data Sets 48
4.3.2 Results 49
4.4 Further Improvements and Suggestions 51
4.5 Conclusion 52
5 Conclusion 53
Bibliography 55
Appendices
A Data format for minconflicts 60
B Data Generator 62
C Test Application 66

List of Tables

3.1 Results obtained for random data using the *lpsolve* MIP solver 32
3.2 Results obtained for Set 1 using *cplex* MIP solver 34
3.3 Results obtained for Set 2 and 4 using *cplex* MIP solver 35
3.4 Results obtained for Set 3 and 5 using *cplex* MIP solver 36

List of Figures

1.1 An example of a forest area divided into stands 5
1.2 A number of schedules as paths in the tree produced by a treatment simulator 7
1.3 Tight integration of simulation in a forest harvest scheduling system 9

2.1 (S1, S2) participating in a neighborhood constraint 18
2.2 Nodes i ,j and k participating in an adjacency constraint 21

4.1 Constraint graph of a forest in Figure 1.1 44
4.2 Function used to calculate height 47
4.3 Variables left inconsistent and iterations (each stand has 5 schedules) 49
4.4 Variables left inconsistent and iterations (each stand has 10 schedules) 50
4.5 Variables left inconsistent and iterations (each stand has 15 or 20 schedules) . 50

Chapter 1

Introduction

Governments and private sectors worldwide manage an immense area of forest. The United States Department of Agriculture alone manages a forest area almost twice the size of Germany. Forest harvest planning and scheduling problems are an important part of this forest management. Recently, both governments and the public worldwide are demanding sustainable harvesting practices for these areas, which would take into account not only economics but also the preservation of biodiversity, and the esthetic, and recreational value of the forest. Long term forest harvest scheduling allows communities, governmental, and non governmental organizations to check if sustainable forestry is actually being practiced. Simplified, the problem is to assign forest treatment types and times to treatment units (often called *stands*) in a given forest area over a long period of time. We will call this problem the Forest Harvest Scheduling Problem (FHSP). An optimization version of the problem involves optimizing objective criteria such as economical, esthetic, and recreational values, which are subject to multiple constraints. Traditionally, mathematical programming models and solution techniques have been extensively used to generate such schedules, and are still the most widely used method (for example, (Garcia 1990, Nelson *et al.* 1991, Yoshimoto *et al.* 1994)). Artificial Intelligence (AI) techniques such as the rule-based expert systems, and the local search, have also been applied to forest management (for example, (Linehan & Corcoran 1994, Eriksson 1994, Murray & Church 1995b)). The above techniques have difficulties in modelling a wide variety of spatial, temporal, and visual constraints and/or in performing the (usually conflicting) multi-criteria optimization that is inherent to the problem. In addition, the size of a real life problem is often beyond the practical capacity of these methods.

Misund *et al.* have suggested the integrated use of Geographic Information Technology (GIT) and certain AI techniques (constraint reasoning and local search) to model and solve such problems (Misund *et al.* 1995). They describe their approach through a simpler version of the problem where only clear-cutting is allowed. The problem is called the Clear-Cut Scheduling Problem (CCSP). CCSP is modelled as a Constraint Satisfaction Problem (CSP) (Tsang 1993) and a solution is iteratively improved using Tabu search (Glover 1989). Although the authors give some reasons for their choice of Tabu search as the iterative improvement technique, it is not all that clear whether the traditional method using a mixed integer linear programming (MIP) model can be effective.

The thesis of this dissertation is that the FHSP (as defined later) can in principle be solved using a MIP model and a state-of-the-art MIP solver if the problem constraints are sufficiently relaxed and if some objective criteria are ignored. However, the size of the problems solvable in a reasonable time is far below the size of any practical real-world problem. Furthermore, an MIP-based strategy can not be easily integrated with the simulation models, which makes the solution hard to implement. The term simulation in our thesis refers to the growth and treatment simulation, which is often used by foresters to find all possible treatment schedules for a stand based on the current stand characteristics and general forestry knowledge. The simulation results can be incorporated into a MIP-based model too but the model formulation alone becomes a demanding task. Therefore, we give a CSP-based model where the simulation can be integrated easily into a solution strategy. Experimental results show that a simple heuristic called *minconflicts* (Minton *et al.* 1992) is able to generate initial solutions that are almost always complete and consistent. The idea is to take these initial solutions and improve them by applying other heuristic optimization procedure like Tabu search. The novel idea in this approach is the integration of the simulation model in a harvest scheduling process using a CSP model.

Our thesis does not solve the full FHSP problem, but rather provides a foundation to start solving practical problems using either mathematical or constraint programming techniques. Our contribution to the FHSP can be outlined as follows. First we conduct a formal study on the restricted version of the problem (CCSP) as given in (Misund *et al.* 1995). We give a MIP model for the CCSP as described later in the problem definition (cf. Section 1.1). We model all of the constraints and the objectives. Evaluating this model on a real-world data shows that it is difficult and time consuming to solve even medium sized problems with all the constraints and the objectives. Next, we integrate

the growth and treatment simulation results into a CSP model which can be used to solve the FHSP. A method for using the simulation results as domain variables for the stands is developed. Lastly, we give an algorithm to generate an initial solution for the FHSP using the *minconflicts* heuristic repair method. Our experimental results show that this method is fast and makes use of the knowledge from the treatment simulation. Therefore, the method can be a good candidate for generating initial solutions to be improved upon later using the iterative techniques. In short, we show that it is possible to model the FHSP as a CSP and use the simulation results so that better forest treatment schedules can be found for the practical problems.

1.1 Problem Definition

Sustainable forest management requires the forest harvesting actions scheduled for a long period of time to obey a variety of constraints. For management purposes, a forest landscape is divided into basic treatment units, often equivalent to *stands*. A stand is a forested area considered homogeneous with respect to a selected set of properties. We will restrict our study to even-aged stands, that is, stands containing trees of the same or almost the same age. We will call the time period of the schedule as *scheduling horizon*. It is a common practice to divide the scheduling horizon into a number of time periods, each period equivalent to, 5 – 10 years.

Definition 1 (Stand) *A stand S_i is an area which is considered homogeneous with respect to certain characteristics. The set of n stands comprises a* forest $\mathcal{F}$, *such that:*

1. $\mathcal{F} = \bigcup_{i=1}^{n} S_i$
2. $S_i \bigcap S_j = \emptyset$, *for all* $i, j, \quad i \neq j$

For each stand S_i,

- $LH(S_i)$: *time of most recent harvest*
- $EARLY(S_i)$: *minimum duration between harvests*
- $LATE(S_i)$: *maximum duration between harvests*
- $OPT(S_i)$: *optimal time between harvests*

- $MH(S_i, H)$: *time required by trees in the stand to grow from 0 to height H*
- $AREA(S_i)$: *area of the stand*
- $GROWTH(S_i, y)$: *volume that may be harvested y years after the last harvest*

Definition 2 (Scheduling Horizon) *A scheduling horizon $\mathcal{H}$ is a contiguous set of m periods $\{P_1, \ldots, P_m\}$, where each P_i is of a certain length in years.*

Definition 3 (Treatment) *A treatment unit (stand) is given a number of management options over the scheduling horizon. We will call individual management option a* treatment. *A set of treatment types $\mathcal{T} = \{T_0, \ldots, T_t\}$ is available for each stand. We let T_0 to be* null treatment *or* "let grow" *where a stand is left without any treatment, and T_1 to be the clear-cut treatment.*

Definition 4 (Forest Harvest Scheduling Problem (FHSP)) *Given an area of forest $\mathcal{F}$ divided into n stands, $\{S_1, \ldots, S_n\}$, find a schedule $\mathcal{S} = \{A_1, \ldots, A_n\}$ over scheduling horizon satisfying a set of constraints $\mathcal{C} = \{C_1, \ldots, C_c\}$. Here, A_i is the set of activities represented by a set of tuples $< P_{ij}, T_{ij} >$, where each tuple represents treatment period and treatment type for the stand S_i. The constraints are usually adjacency constraint, harvest interval constraint* et cetera.[1]

Now we describe the constraints and the optimization criteria involved in the FHSP by defining the Clear-Cut Scheduling Problem (CCSP) in detail [2]. The CCSP is a special case of the FHSP where (besides null treatment) only one other treatment (clear-cutting) is allowed, and each stand is assigned only one management option over the scheduling horizon. Figure 1.1 is an example of a forest area divided into stands. Two stands sharing a common border are defined as *adjacent*, or *neighbors*.

Definition 5 (Clear-Cut Scheduling Problem (CCSP)) *The Clear-Cut Scheduling Problem is a restricted FHSP. Only two possible treatment types are allowed, T_0 and T_1 ("let grow" and clear-cut). The clear-cut treatment is scheduled only once for each stand during the entire scheduling horizon.*

[1] The constraints will be described later

[2] This closely follows the definition given in (Misund *et al.* 1995).

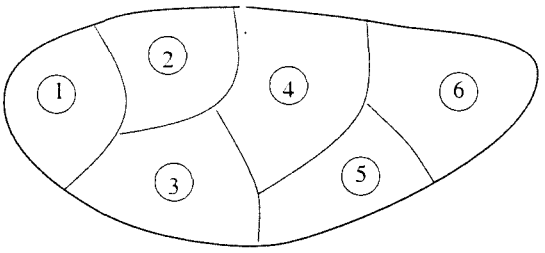

Figure 1.1: An example of a forest area divided into stands

A CCSP can be visualized as assigning values to each of h_i, $1 \leq i \leq n$, where h_i is the scheduled harvesting (clear-cut) period for the stand S_i, rest of the periods are assigned the null treatment. The value of each h_i is picked from its domain $D_i = \{1, \ldots, m\}$ where m is the total number of periods, such that the problem constraints (defined shortly) are satisfied. We define the constraints and the optimization criteria for the FHSP as follows.[3]

Constraints Constraints can be generally divided into two categories - hard and soft. The hard constraints are defined as those constraints that must be satisfied whereas the soft constraints can be relaxed to satisfy the hard constraints or to adjust the objective function value.

Hard Constraint C_1: Minimum height constraint All the neighbors of the stand to be harvested must have an average tree height of at least X meters. For CCSP,

$$\forall i \in \{1, \ldots, n\} : \forall j \in I(i) : (h_i < h_j - MH(S_i, X)) \bigvee (h_i \geq h_j + MH(S_j, X)) \quad (1.1)$$

where $I(i)$ is index set for neighbors of S_i.

Soft Constraint C_2: Harvest Interval Preferred harvest interval based on economical and ecological considerations. A lower and upper threshold is given for each stand, as well as the optimal harvest time relative to the last clear-cut. For CCSP,

$$\forall i \in \{1, \ldots, n\} : EARLY(S_i) \leq h_i - LH(S_i) \leq LATE(S_i) \quad (1.2)$$

[3]These are taken from forest harvesting requirements for the Norwegian ECOPLAN project (Misund *et al.* 1995) but are also general enough to be applicable to forest harvesting practices elsewhere.

Optimization Criteria A *feasible solution* is an assignment of problem variables such that all the constraints are satisfied. An optimization version of the CCSP (and the FHSP) arises when various optimization criteria are maximized (or minimized) in the solution. We are interested in solving the optimization version of the problem with the following optimization criteria:

O_1: **Optimal Harvest Time** The actual time between harvests should be as close to the optimal as possible. For CCSP,

$$Minimize \quad \sum_{i=1}^{n} (|OPT(S_i) - (h_i - LH(S_i))|) \tag{1.3}$$

O_2: **Even Harvest Volume** The estimated harvested volume $EHV_p(\mathcal{S})$ for any period p should be as close to the average harvested volume in a schedule $\mathcal{S}$, $AHV(\mathcal{S})$. In other words, the variance between the harvest volumes should be minimized.

$$Minimize \quad \sum_{p=1}^{m} (|EHV_p(\mathcal{S}) - AHV(\mathcal{S})|) \tag{1.4}$$

O_3: **Old Forest** A specified minimum area of old forest (AOF), that is, the total area of the stands with average age above a certain threshold, should be maintained over the scheduling horizon. Let $OLD(p, \mathcal{S})$ be the area of old forest in period p.

$$Minimize \quad \sum_{yp=1}^{m} max(0, AOF - OLD(p, \mathcal{S})) \tag{1.5}$$

Note that the optimal would be 0 when the area of old forest is greater than or equal to AOF.

O_4: **Visual Impact** Minimize the visual damage of clear-cutting relative to a set of viewpoints. Let $VIS(S_i, p, V)$ denote visual impact from stand S_i in period p relative to viewpoint V. Note that the following equation only assumes one viewpoint, whereas in the real case there is usually a set of viewpoints.

$$Minimize \quad max_{p=1}^{m}(\sum_{i=1}^{n} VIS(S_i, p, V)) \tag{1.6}$$

The *minimum height constraint* is enforced so that a large area of forest is not clear-cut simultaneously causing a great damage to the regeneration process and the wildlife habitats.

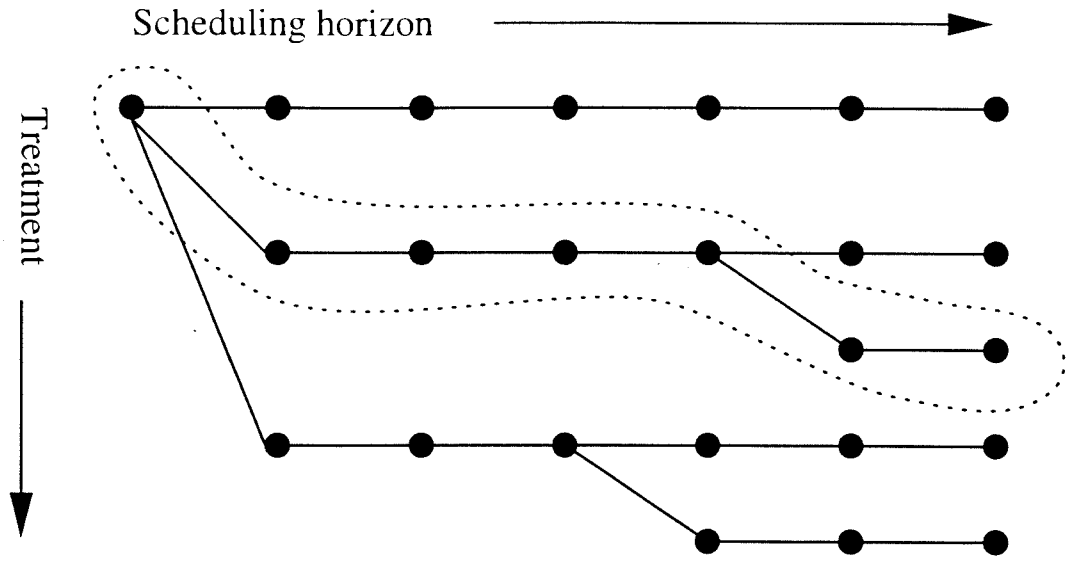

Figure 1.2: A number of schedules as paths in the tree produced by a treatment simulator

It is also referred to as *adjacency constraint* because cutting adjacent stands at the same time period is prohibited by the constraint. This restriction is usually valid for certain specified periods such that the harvested area will have regenerated properly. These periods are commonly called *exclusion periods* or *green-up age*. Recently, the adjacency constraint and the exclusion period have been given special consideration in the majority of research dealing with the forest harvest scheduling (according to our review in the next section).

1.1.1 Growth and Treatment Simulation

In this thesis, the term simulation refers to the growth and treatment simulation of areas of forest as practiced and defined in the forestry literature (Hoen 1992, Lappi 1992). Growth simulation is usually implemented in a simple way using growth curves for a variety of tree species. On the other hand, treatment simulation is more involved and requires more forestry knowledge and stand characteristics data. Various factors come into play when generating an allowable treatment for a given stand. Although the simulation assumes that each stand is independent from the others, a state of the stand changes each time a treatment is applied. Depending on this state, future treatments are simulated.

Figure 1.2 is a small example of the growth and treatment simulator result for a stand. The simulator in effect produces a tree where each node represents a period in the scheduling horizon, which is increasing in the X-axis. Different treatments are represented in the Y-axis at every branch of the tree. For example, the first level could be the "let grow" treatment or do nothing, the second clear-cut, the third commercial thinning and so on. Every path in the tree amounts to a complete schedule for the stand. Thus, the number of leaf nodes in the tree is equivalent to the number of alternate schedules available for the stand.

It is evident that any harvest scheduler that integrates simulation results produces a correct and implementable schedule. The problem is that each tree as in Figure 1.2 can have hundreds of leaf nodes. For efficiency, only some of the leaf nodes can be selected as the possible schedules during the scheduling phase. There has not been any published work (to our knowledge) that tries to integrate the simulation model into the scheduling process in such a way that the problem constraints are also satisfied simultaneously. Almost all of the scheduling systems to date do not take the treatment simulation into account while generating harvest schedules. Another problem with such simulation is the independence-of-stands assumption. A scheduler has to take care of all the conflicts that may arise from the constraints between the stands such as the adjacency constraint. Thus, the scheduler will perform better if every stand has a number of alternate schedules.

Our goal is to integrate the growth and treatment simulation into the harvest scheduling process. We study whether this can be done with either the mathematical or the constraint programming. Ideally, we would like to have an initial solution that uses the simulation results and the optimizer improves the solution with the help of the simulator. This is illustrated in Figure 1.3. It is a *tight integration model.* If the simulator results are compiled beforehand and given to the optimizer, then we have a *loose integration model.* In our thesis, we only deal with the loose integration model.

1.2 Review of Optimized Forest Harvest Scheduling

There are two major classes of forest harvest scheduling algorithms. One is based on mathematical programming and the other on heuristic techniques (with or without using simulation in both cases). The former is a global procedure which finds an optimal solution to the forest management model whereas the latter is usually a local procedure which iteratively improves the solution without any guarantee of finding the optimal one. Mathematical

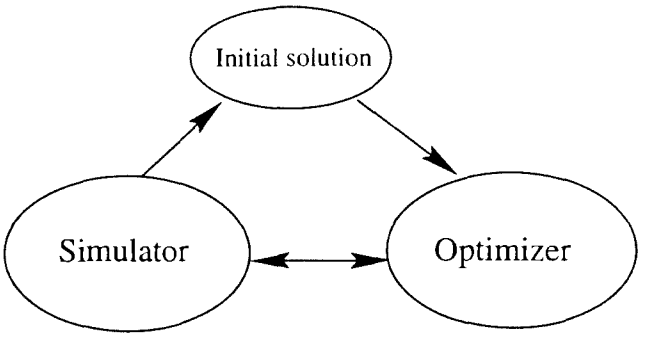

Figure 1.3: Tight integration of simulation in a forest harvest scheduling system

programming is the technique that is commonly used in practice and consequently a large percentage of the research activity is devoted to it. There are also some algorithms which do not fall under these two categories, and that we will discuss under the miscellaneous methodology.

1.2.1 Mathematical Programming

The general forest planning problem, where harvesting is an important process, has been studied for some time. Early forest management models (Navon 1971, Johnson & Crim 1986) were developed using Linear Programming (LP). Two well known LP-based scheduling packages in use are FORPLAN (Johnson & Rose 1986) and MUSYC (Johnson & Jones 1979). Johnson and Scheurman reviewed and analyzed many LP-based forest planning systems (Johnson & Scheurman 1977). In this paper, the authors group LP models into Model I and Model II, which are frequently used in forest planning. Later Garcia in his review of LP in forest planning (Garcia 1990) revises the classification. All these models are geared towards LP-based solution strategies.

LP models use continuous variables. Defining spatial relationships requires the use of integer decision variables. For example it is not possible to capture the adjacency constraint just by using continuous variables. Thus, LP-based model are aspatial models. These models therefore have been used at strategic planning level. But implementing the plan at an operational level, that is, designating a spatial area of forest for harvesting, is very difficult and may be impossible. Furthermore, the solution obtained from such a model is

nonintegral. Consequently, LP-based solutions are difficult to interpret. For example, how informative would be a solution suggesting treatment 2.3 for a stand or period 1.5?

Since LP-based models are not able to express spatial relationships, the research began to shift towards the mixed integer programming (MIP) models (Kirby *et al.* 1986, Jones *et al.* 1991). An integer decision variable is used to express a particular harvesting decision. This allows the model to express spatial relationships in the form of adjacency constraints. In Chapter 3, we will give a MIP model for the CCSP such that the spatial relationships are captured in the constraints. Since the general integer programming algorithms are not efficient when a large number of integer variables exist, the MIP-based strategies are restricted to small problems. One way of making the MIP models efficient is using an improved representation of adjacency constraints (Meneghin *et al.* 1988, Torres-Rojo & Brodie 1990, Jones *et al.* 1991, Yoshimoto & Brodie 1994). Researchers have been steadily solving larger and larger MIP problems using the improved constraint formulations and a variety of heuristic algorithms. Weintraub *et al.* solved a MIP forest harvesting model with adjacency constraints for multiple time periods using linear programming, and a sophisticated rounding heuristic (Wientraub *et al.* 1994). The LP relaxation of the MIP model is solved and passed to a MIP solver which attempts to assign integer values using the heuristic algorithm.

The LP models are sometimes used together with a growth and treatment simulator. The simulator is used to find all possible treatment schedules for all the stands. In most cases, each stand is treated independently and the necessary information is provided as the simulation proceeds. In LP models all the information has to be encoded beforehand in the model. Some researchers have tried to combine these two techniques (GAYA-LP (Hoen 1992) and JLP (Lappi 1992) are good examples). In both cases, the growth and treatment simulator is used to define all the allowable treatments for each stand and the output is optimized by a LP solver. The net present value of the forest is often used as the optimization criteria. It is calculated using various economic and forest data such as the interest rate, the volume harvested, the species of the harvested trees *et cetera.*

1.2.2 Heuristic Optimization

There have been several studies exploring the heuristic optimization techniques for the FHSP with (or without) mathematical programming. One of the approaches considered successful is the sampling heuristic called Monte Carlo Integer Programming (MCIP) (Nelson & Brodie 1990). It is a biased sampling scheme that generates a number of feasible

solution alternatives. The solution becomes better as the number of samples increases. Therefore, the optimal or the near optimal solution may be possible only if a very large number of samples are generated. Unfortunately, this increases significantly the time necessary to find a solution. Lockwood and Moore use Simulated Annealing (SA) to generate harvest schedules with spatial constraints (Lockwood & Moore 1993). SA is a stochastic optimization technique that has been successfully used to solve combinatorial optimization problems (Kirkpatrick 1983, Kirkpatrick 1984). The authors report to have solved large harvest scheduling problems with adjacency constraints only. Briefly, annealing is a natural phenomenon (for example in metals) in which the internal elements of a cooling body rearranges their order from a high to a low energy state. From the molten (freely moving) state the body cools down slowly to a stable state with minimal energy. If it is cooled too fast the body hardens into a suboptimal state. SA tries to imitate this phenomenon by gradually rearranging the elements of a system from disordered to ordered state. If the system is cooled slowly enough, SA guarantees (probabilistic) convergence to an optimal solution.

A recent study compares three heuristic approaches to solving the forest planning problems, of which harvest scheduling is a part (Murray & Church 1995b). The authors model the problem as a MIP which allows the representation of the adjacency constraints. The only objective used in the study is the maximization of the net revenue. They develop a method which improves upon the solution produced by Monte Carlo sampling process using Hill Climbing (HC), Simulated Annealing (SA), and Tabu Search (TS). In all the three approaches, the initial solution produced by Monte Carlo sampling is locally improved by generating new neighbourhood solutions. In HC, only an improved solution is accepted in every step and therefore is more likely to get stuck in a locally optimal solution. When minimizing a locally optimal solution is called *local minimum.* These local minima hinder the heuristic from moving towards the globally optimal solution. SA and TS accept a worse solution (with some probability) in hope of escaping the local minima. These methods were tested using the data from (Nelson & Brodie 1990). It was found that TS performed better more often than SA or HC. However, this does not imply that TS will always produce a better solution than the other two approaches. The authors confirmed this using Friedman analysis (Conover 1980) on the solution results. Given any initial solution, it is equally likely that a high quality solution be reached by any one of the three techniques.

Pukkala and Kangas describe another heuristic optimization technique (Pukkala & Kangas 1993). Their method uses a growth simulator as the first step to produce several alternative treatment schedules for each stand. The second step is the actual heuristic optimization, in which the total utility of the schedule is maximized by randomly picking a treatment schedule, where the utility is calculated by adding the values of the objective function. This method was tested on a data of a small forest area and was found to be successful. It is claimed to be better than the LP-based methods because of its ability to express nonlinear objectives. The drawback of this method, however, is that it does not take take into account the adjacency constraints. Therefore the schedules generated may be of very poor quality.

Constraint Satisfaction Problems (CSPs)

Almost all of the methods mentioned above have an underlying mathematical programming model for solving the forest harvesting problem. One interesting study is (Misund *et al.* 1995), where the problem is modelled as a constraint satisfaction problem (CSP) for the first time (to our knowledge). The authors suggest the CSP model and an iterative improvement technique, such as TS, to solve the FHSP. They used the restricted version of the problem (CCSP) as a test case. We will study this case in more detail in Chapter 2 and Chapter 3.

There are well-studied AI search methods that can be used to solve CSPs (Tsang 1993). These methods can be classified into constructive and iterative search methods. Constructive search, for example standard backtracking (Bitner & Reingold 1975), starts with a particular ordering of the variables and instantiates them one at a time. Thus it works with a partial solution and tries to extend it to a full solution. This search method suffers from a phenomenon called *thrashing* whereby the same variable-value pair that leads to no solution is instantiated over and over again during the search process. These algorithms have exponential time complexity. However, this search technique is complete, that is all the possible solutions can be found, and the optimal one identified.

The alternative to the constructive search is to start with an initial solution and use a local search heuristic (for example, Tabu search (Glover 1989) or simulated annealing (Kirkpatrick 1984)) to iteratively improve the solution. This method has been shown to be much more effective for large and complex search problems. The advantages of using such methods are that they are fast for some problems, that the current best solution is available any time, and that fairly large problems can be attempted. The drawback is that the search can get stuck in local optima or minima and consequently finding the global optimal solution

can become impossible.

In addition to a search technique, a solution strategy for the CSPs can also consist of a consistency enforcement mechanism, for example, *arc-consistency* (Mackworth 1977). These algorithms reduce the size of the search space by eliminating those parts that can not contain a solution. Therefore, a typical solution strategy for a CSP interleaves a search mechanism with the consistency enforcement.

1.2.3 Miscellaneous Methodology

There are a few other studies that do not fall under the above categories. One study uses 0-1 integer programming to determine patterns for forest harvesting with adjacency restrictions and forbidden regions modelled as a grid-packing problem (Snyder & ReVelle 1996). Another recent study suggests a method based on Bayesian statistical concepts (Van Deusen 1996). Still these methods do not solve the FHSP problem as we defined it, especially the multi-criteria optimization part.

Other techniques are those based on control theory, and on non linear and dynamic programming (Roise 1986, Hof & Joyce 1992). Even though many objectives in the FHSP can be written as nonlinear constraints, there has been very little work done in this area. Most of the nonlinear programming research deal with the problem of habitat scheduling along with the harvest scheduling. However, the size of the problem that is actually being solved is very small because of the additional habitat constraints and the inefficiency of general nonlinear program solvers.

1.2.4 Summary

Most of the methods mentioned before in the review have been tested with very small problems compared to the practical ones that exist. Many studies use only 5–7 periods and 100 stands whereas any real-life problem is bound to have stands in the order of thousands and approximately 20 periods (usually of length 5 years). The problem size is kept small because MIP models tend to consist of a large number of binary integer variables (or integrality constraints). These constraints restrict the size bounds for problems because of the limitations of general integer programming solution procedures. For example, in one study considering only adjacency relations the number of constraints is $\mathcal{O}(nm)$, where n and m are the number of stands and periods respectively (Murray & Church 1995b). The number

of binary variables needed to formulate these constraints is nm. Furthermore, each such variable participates in a number of constraints. A practical problem typically has at least 5000 stands and 20 periods. That is, 100.000 integer variables – a large number for current MIP solvers.

However, some studies have developed and added various heuristics to solve increasingly larger harvest scheduling problems. Furthermore, in most of the experiments the scheduling horizon is kept very small to keep the number of binary variables (and thus the integrality constraints) under control. But these methods do not deal with all the constraints and the objectives simultaneously as we have illustrated in the problem definition (Section 1.1). Therefore, a MIP model which tries to minimize the number of binary integer variables using efficient constraint formulations and/or good cut constraints is needed. These cut constraints help produce an integral solution faster. We will study some of the cut constraints relevant to our problem in the first part of Chapter 2.

Most of the algorithms we reviewed implements growth and yield simulation using growth and yield curves for a variety of tree species. But the treatment simulation is almost never used. We reviewed some research which tried to integrate the treatment simulation with a LP solver to find harvest schedules (Hoen 1992, Lappi 1992). However, the problem with these approaches is the LP model since not even the adjacency constraint (spatial relations) can be represented in the model. Integrating the treatment simulation with a MIP has not been done so far and we will see in Chapter 3 that this may be possible if we can utilize column generation technique. However, a MIP model formulation for this is a challenging task. Therefore, an alternative model is needed, in which the integration of the simulation would be easier than in the MIP models. A constraint satisfaction problem (CSP) model is introduced in (Misund *et al.* 1995). The authors integrate constraint reasoning techniques with geographical information technology (GIT). GIT is helpful in visualizing the solution to explore the possibillity of further improvements. This is effective for example regarding the visual effect of a particular schedule. It is therefore imperative that models be built such that these techniques can also be exploited properly. The CSP model is attractive for this purpose since new models can be integrated easier than in a MIP. We suggest this model for integrating the simulation and the scheduler for the FHSP.

1.3 Overview of the Thesis

The rest of the thesis is organized as follows. Chapter 2 describes a formal study on the structure of the CCSP to gain a deeper understanding of the problem. Good cut constraints (defined in the next section) are needed to solve a MIP model for integer solutions effectively. We identify a constraint formulation in the CCSP and explain why it is useful during the MIP solution process. Studying the adjacency constraint in the CCSP reveals that the problem is very loosely constrained. It means that preprocessing the problem using consistency techniques (Mackworth 1977) is not worthwhile. Consistency enforcement is a technique commonly used while solving the CSPs. Furthermore, it implies that satisfying just the adjacency constraint in CCSP is easy. This does not necessarily say anything about the hardness of the whole problem when the objective criteria are also included. In Chapter 3, we give a MIP model for the CCSP as described in Section 1.1. The adjacency constraints are formulated in such a way that they behave as the cut constraints. We also outline some empirical evidence showing that small-sized problems with some constraint relaxations can be solved using the model. This exercise also shows that solving practical problems using currently available MIP solvers is unrealistic.

Since it is unlikely to solve the full FHSP using just the MIP techniques, we give a model based on the CSP model. We also integrate the growth and treatment simulation in this model. Based upon the research to date, our reason for doing this is twofold. Firstly, MIP *can* give solutions but the solvable size of the problems is very small compared to the practical problems that exist. Secondly, incorporating simulation results into a MIP would be difficult and the problem decomposition techniques (for example, the column generation) have to be figured out first. This type of problem is most likely solved using some sort of iterative improvement technique together with a wide variety of available heuristics. For these techniques to work well, it is believed that the initial solution has to be reasonably "good". We therefore present a method of generating an initial solution that uses the known *minconflicts* heuristic (Minton *et al.* 1992). Our measure for goodness is the number of consistent variables in the initial solution. Our experimental results show that this method can be used to generate consistent solutions.

Finally, we give concluding remarks and some future research directions in Chapter 5.

Chapter 2

Formalization of the FHSP

The Forest Harvest Scheduling Problem (FHSP) is a complex problem involving spatial and non-spatial constraints and (conflicting) objective criteria (cf. Chapter 1.1). It is an example of a hard combinatorial optimization problem. The search space for these problems is huge. To illustrate, let us take an example of the CCSP (only considering the adjacency constraint) with 500 stands and 20 periods. Then the total number of schedules, both feasible and infeasible, is 20^{500}, which is more than the number of atoms in the universe! Furthermore, the optimization criteria add complexities to the CCSP. Such complex problems can be solved effectively only if problem-specific heuristics are developed that searche only part of the search space. Another approach is to reduce the size of the search space itself by eliminating those parts which do not contain any feasible (or optimal) solution.

The problem solving techniques we are going to study are mathematical programming, MIP in particular, and constraint programming using a CSP model. To solve a MIP model for an integral solution, we need an efficient constraint formulation and we have to identify good cut constraints (defined shortly). Efficient constraint formulations help reduce the number of integer variables in the model and the cut constraints help quicken the MIP solution process by converging to the integral solution faster. We study the CCSP structure to identify these formulations. A constraint solving technique is more effective if the search space is reduced to a manageable size. However, the search space is huge even for the CCSP which is already a restricted version of the FHSP. Nevertheless, it is usually the case that a large area of search space can be thrown away by enforcing consistency techniques, for example arc-consistency. We study the problem structure to see if such techniques can be used effectively to solve the CCSP (and the FHSP).

We first study the CCSP problem structure to identify the efficient constraint and cut constraint formulations for a MIP model. Then we study constraint structure in the CCSP to see if any of the consistency enforcement techniques can be applied to speed-up the solution process.

2.1 Identifying Cuts

In this section we will study the relaxed version of the FHSP problem (CCSP), mainly its structure and the constraints. We study the node-packing problem and show that the CCSP consists of many such problems as sub-problems. We study the adjacency constraint structure in particular. First, some definitions are required.

Definition 6 (Set-packing Problem (Nemhouser & Wosley 1988)) *Any problem that can be defined as linear system of inequalities* $\mathbf{Ax} \leq \mathbf{b}$, *such that* $\mathbf{A}$ *is a* $(0,1)$ *matrix,* $\mathbf{b}$ *is an unit vector and* $\mathbf{x}$ *is binary.*

Definition 7 (Node-packing Problem) *Given a graph* $G = (V, E)$, *where* V *is the set of vertices and* E *is the set of edges, find* $U \subseteq V$, *and* $|U| \geq 2$, *such that no pair of nodes in* U *is joined by an edge.*

Definition 8 (Branch-and-Bound) *It is the most popular integer programming algorithm. The original problem is divided into smaller and smaller sub-problems. The dividing (*branching*) is the process of partitioning the entire set of solutions into smaller and smaller subsets. Then, the best solution in each subset is* bounded *and the subset discarded if the bound indicates that it can not contain optimal solution for the original problem. The discarding process is also called* fathoming.

Definition 9 (Cutting plane (Cut)) *A cutting plane (cut) for an integer program is a new constraint that eliminates some feasible solution of the original LP relaxation while keeping every integral feasible solution intact. Cuts tighten the bound obtained in the bounding step of the branch-and-bound technique and improves its performance.*

Some researchers have shown that a binary integer (0-1) model for the node-packing problem has better chance of converging into an integer solution in practice (for example, (Gebotys & Elmasry 1993, Hoffman & Padberg 1991)). Gebotys and Elmasry apply a

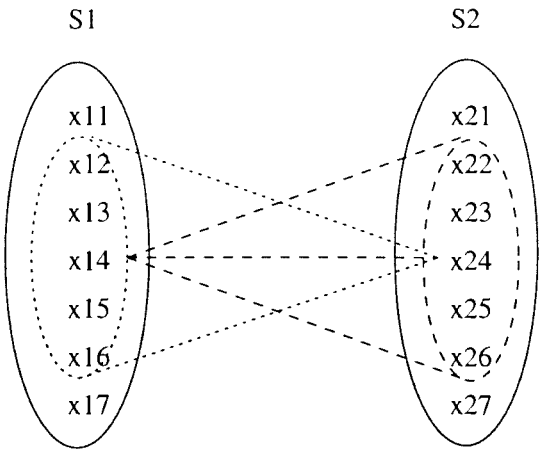

Figure 2.1: (S1, S2) participating in a neighborhood constraint

node-packing problem representation for architectural synthesis problems. Architectural synthesis is an important part of the VLSI design cycle. The objective is to transform the input algorithm into a hardware architecture that minimizes a cost function while satisfying a set of constraints. They model this problem as a 0-1 integer programming problem and solve the previously unsolved instances. Their success can be attributed to the constraint formulations which act as good cuts during the branch-and-bound algorithm for integer solutions.

We now model the CCSP (cf. definition in Chapter 1.1) as a 0-1 integer program. We start with a binary (0,1) variable x_{ij} for each stand S_i and each period p_j, where $i = 1 \ldots n$ and $j = 1 \ldots m$. That is, if $x_{ij} = 1$, then stand S_i is harvested in period j.[1] Now our problem is to assign integer values to all x_{ij} such that the CCSP constraints are satisfied. We can represent this problem by a system of inequalities $\mathbf{Ax} \leq \mathbf{b}$, where the rows of $\mathbf{A}$ correspond to the coefficients (values) $[x_{11} \ldots x_{1m}, x_{21} \ldots x_{2m} \ldots x_{n1} \ldots x_{nm}]$, where $\mathbf{x}$ is binary and $\mathbf{b}$ is a unit vector.

$$[x_{11} \ldots x_{1m}, x_{21} \ldots x_{2m} \ldots x_{n1} \ldots x_{nm}] \quad \mathbf{x} \quad \leq \quad \mathbf{b}$$

Thus, by definition this is a set-packing problem. Set packing problems can be transformed into a node-packing graph G(V,E) (Nemhouser & Wosley 1988). Therefore, the CCSP can

[1] It is standard practice in forestry literature (cf. Chapter 1)

be transformed into a node-packing graph G(V,E).

In Figure 2.1, two stands (S_1, S_2) are neighbors and it is assumed that the time for trees to grow to the minimum height of say 2 meters, MH, is 2 periods and the length of scheduling horizon is 7 periods. Then, we immediately see that

$$\sum_m x_{im} \leq 1, \ \forall i.$$

That is, a stand is harvested only once [2] in the scheduling horizon. The solid line ellipsoids in Figure 2.1 group the variables participating in these constraints. Another constraint of this kind that becomes apparent is the *minimum height* constraint. For example, in Figure 2.1 if stand S_1 is harvested in period 4, then $x_{14} = 1$, which means that stand S_2 can not have been harvested 2 periods before and can not be harvested 2 periods after the period 4. We can represent this in general by the equation:

$$x_{it} + \sum_{(t-MH) \ mod \ m \leq n \leq (t+MH) \ mod \ m} x_{jn} \leq 1, \ \forall \text{ neighbors } (S_i, S_j).$$

We can graphically view this equation in Figure 2.1 as dotted cones. This equation is symmetric for the stands S_1 and S_2 since we are assuming only one treatment per stand in the CCSP, it can not be the case otherwise. This is a subgraph of the node-packing graph. For example, in Figure 2.1, $V = \{x_{ij}\} \ \forall i \in \{1, \ldots, n\}$ and $\forall j \in \{1, \ldots, m\}$ and $E = \{(X_i, X_j) \ | X_i \text{ and } X_j \text{ can not be simultaneously 1}\}$. Then we can choose $u_1 \in S_1$ and $u_2 \in S_2$ where $u_1, u_2 \in U$, and that would be our solution. Note that it is trivial to solve node packing in this case, just choose x_{1j} from S_1 and choose the node x_{2k} from S_2 such that $|j - k| > MH$. However, it may not be so simple if S_1 or S_2 is also adjacent to some other stand.

2.1.1 0-1 Node-Packing Problem

We noted earlier that branch-and-bound is a technique for solving general integer programming problems. It is an efficient technique in the sense that the algorithm identifies and ignores the feasible solutions which can not be the optimal one. Nevertheless, it is still restricted to problems with a small number of integer variables due to the very large number of the feasible solutions which may exist. General 0-1 integer program is NP-complete (Nemhouser & Wosley 1988). However, in some special cases, polyhedral theory can be applied

[2] It is an assumption in the definition of the CCSP

to solve them more efficiently. Any bounded system of linear inequalities can be visualized as a *polytope* such that the solution(s) can only be the vertices of the polytope. An *integral polytope* is a polytope where all the vertices that are the solutions are also integral. It has been proven that for every polytope there exists an integer polytope that can be solved as a linear program and always produce an integer solution (Nemhouser & Wosley 1988). All integral *facets* of the polytope are necessary to be formulated as the constraints to identify its corresponding integer polytope, where a facet is a hyper-plane of dimension one less than that of the polyhedron.

There exist problems for which all such integral facets are characterized, for example, matching and total unimodularity (Nemhouser & Wosley 1988). But for the node-packing problem, it is only partially characterized. Finding all integral facets for the node-packing problem is NP-complete (Gebotys & Elmasry 1993). It is still worthwhile finding some of these facets because they make the model tight, that is, reduce the size of the solution space, and give good bounds for the problem. This in turn is expected to make a branch-and-bound algorithm more effective.

In graph-theoretic terms, a set $C \subseteq V$ is called a *clique* if every pair of nodes in C have an edge. In other words, C is a complete graph. Therefore, in a node-packing, only one node from a clique is allowed in the solution. Note that the ellipsoids in Figure 2.1 are cliques. A clique is *maximal* if it can not be contained in any other cliques. Each maximal clique in a node-packing problem can be formulated as a *clique constraint* or *clique inequality.* The clique constraint is a known integral facet of the node-packing problem (Nemhouser & Wosley 1988). Therefore, it is in our interest to find all the maximal cliques in the problem. These inequalities will not define all the integral facets. However, it is likely that they will reduce the number of branch and bounds later while solving for optimal integer solution.

Lemma 1 *The cliques represented by the ellipsoids in Figure 2.1 are maximal assuming* $MH \geq sizeoftheschedulinghorizon.$ *Let a window for* x_{ij} *of stand* S_{ij}, W_{ij}, *be the set of nodes in* S_j *that can not be simultaneously one with* x_{ij}, *whenever* S_i *and* S_j *are neighbors. The clique represented by the cone for every variable* x_{ij} *in Figure 2.1 is also maximal if* W_{ij} *is not contained in any other window for the variables other than* x_{ij}.

Proof: Note that the cliques represented by the bigger ellipsoids are maximal. The ones represented by the dotted cones are maximal because all of them are unique. This is because each x_{ij} together with the x_{jk} where k is a window, W_{ij}, of interval $[(t - MH)\ mod\ m, (t +$

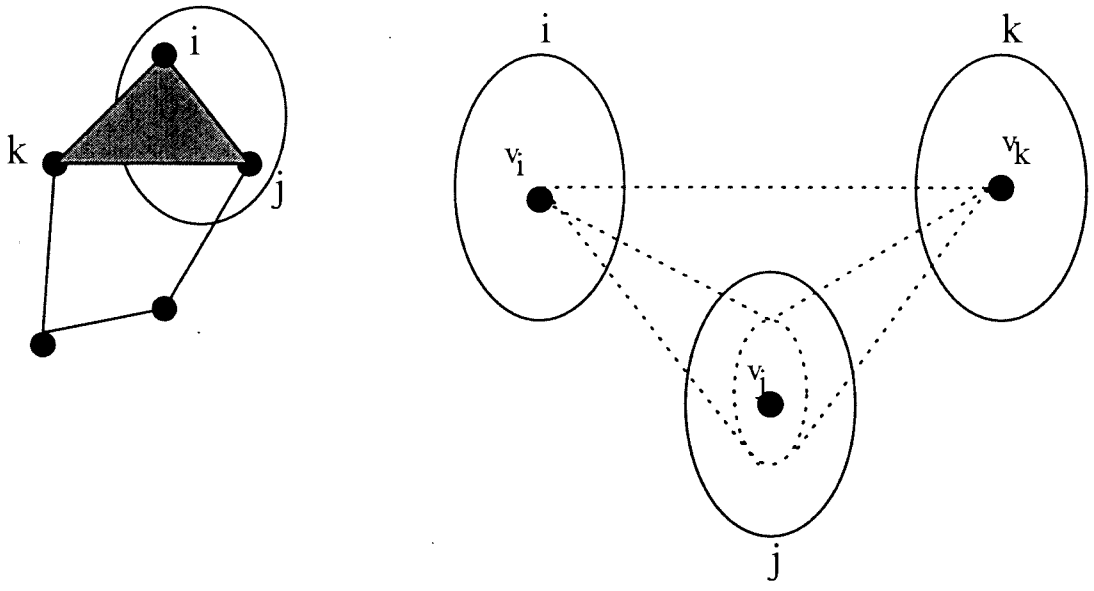

Figure 2.2: Nodes i ,j and k participating in an adjacency constraint

$MH)$ *mod* m] forms a clique. If this is not contained in any other window for variables other than x_{ij}, then it is unique and hence the clique is maximal. ■

Note that this may not be the case when we consider the complete problem. There may exist a number of structures similar to Figure 2.1 as in Figure 2.2, where the nodes i, j and k are part of separate cliques. We can call the graph on the left of the figure an *adjacency graph*, where the nodes are stands and edges represent the adjacency constraint. Let v_i, v_j and v_k be one of the binary variables for the nodes i ,j and k in respective order. Then $v_i + (v_j + MH_i) \leq 1$; $v_i \neq v_k$; $v_k + (v_j + MH_i) \leq 1$. Thus, what used to be a maximal clique for any two nodes does not remain so any longer because it can be extended. For example, the clique represented as the cone for nodes i and j can be extended with another node v_k, which also shares the same variables from the node j to form a clique.

To use clique constraint as integral facet, all the maximal cliques in the original adjacency graph has to be determined. [3] Since the graph is planar, size of the maximal cliques is bounded by 4. All these cliques can be found by looking at all possible combination of three and four vertices in the graph. Thus, the complexity is of $\mathcal{O}(n^4)$, where n is the number

[3] Not to be confused with the maximal cliques of binary variables

of vertices in G. Another known integral facet for node-packing is $\sum_{i \in C} x_i \leq \frac{(|C|-1)}{2}$, for all chordless cycles C, $|C| \geq 5$. Finding these facets for large problems can lead to integral solutions faster.

Lemma 2 *For every pair of neighbors (S_i, S_j), it is sufficient to include in the MIP model the clique inequalities defining the adjacency constraint from only one stand.*

Proof: It is easy to see that the constraints are symmetric and therefore, one set of constraints originating from S_i also essentially captures the set of constraints originating from S_j. ■

Using Lemma 1 and 2, we expect that the chances of finding integer solutions to our model with reasonable number of constraints are higher. We describe our MIP model in detail in Chapter 3.

2.2 Consistency Enforcement

The FHSP can also be represented as a constraint network (defined below). There are various algorithms that eliminate parts of the search space for such networks by throwing away inconsistent solutions. One such algorithm is arc-consistency. However, these algorithms are wasteful if the problem (constraint network) is loosely constrained. Therefore, we want to find out how "loose" our problem is. To formally study this, we adopt van Beek's concept of *constraint looseness* which gives the lower bound on the inherent level of local consistency in a binary constraint network (Van Beek 1994). For graph coloring problems, these bounds are tight.

Definition 10 (Binary Constraint Network (Montanari 1974)) *A binary constraint network consists of a set X of n variables $\{x_1, x_2, \ldots, x_n\}$, a domain set D_i of possible values for each variable, and a set of binary relations between variables. A binary relation or binary constraint, R_{ij} between variables x_i and x_j is a subset of cartesian product of their domains. An instantiation of variables in X is a n-tuple $\{X_1, X_2, \ldots, X_n\}$ such that $X_i \in D_i$ is assigned to x_i. A consistent instantiation of a network, also called* solution, *is an instantiation of variables such that all the constraints between variables are satisfied.*

Definition 11 (Strong k-consistency (Freuder 1978, Freuder 1982)) *A network is k-consistent iff given any $k-1$ consistent instantiations of variables there exists another*

consistent instantiation for the k^{th} variable such that all the k instantiations are consistent with one another. If $k = 2$ or $k = 3$, the network is called arc-consistent *or* path-consistent *respectively. A network is called* strongly k-consistent *if and only if it is j-consistent for all $j \leq k$.*

Definition 12 (m-looseness (Van Beek 1994)) *A binary constraint is m-loose if every row and every column of the (0,1)-matrix that defines the constraint has at least m ones, where $0 \leq m \leq |D| - 1$. A binary constraint network is m-loose if all its binary constraints are m-loose. The (0,1)-matrix representation of the constraint between variables x_i and x_j is given by*

$$R_{ij,ab} = \begin{cases} 1 & \text{if } a \neq b \wedge |a - b| \neq |i - j| \\ 0 & \text{otherwise} \end{cases}$$

Theorem 1 ((Van Beek 1994)) *If a binary constraint network is m-loose and all domains are of size $|D|$ or less, then the network is strongly $\left(\left\lceil \frac{|D|}{|D|-m} \right\rceil\right)$-consistent.*

The above definitions and the theorem can be understood in terms of a graph coloring example, say k-coloring problem. Then, the problem is k-consistent. It is also $k - 1$ loose because every binary constraint between two variables disallows the assignment of the same colors for both. Therefore, the problem is $\frac{k}{k-k+1}$-consistent, that is, k-consistent. Above definitions are generalizations of this observation.

Representing node-packing constraint from Figure 2.1 in $(0, 1) - matrix$ form gives the following matrix,

$$R_{ij} = \begin{vmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{vmatrix}$$

Note that for any row, the maximum number of zeroes is $2(MH - 1) + 1$, that is, in any row i, the columns that are $(MH - 1)$ to the left and the right and the i^{th} column itself can be zero. Therefore, the minimum number of ones is bounded by $|D_i| - 2(MH - 1) + 1$. The matrix is diagonally symmetric. Thus, same number is also the lower bound for the columns. It is usually the case that in a practical FHSP $|D_i|$ is much greater than MH. For example,

$|D_i| = 20$ and $MH = 5$. Thus, this network is $m - loose$, where $m = |D_i| - 2(MH - 1) + 1$ or 13. Then, according to Theorem 1, the network is strongly 3-consistent. Thus, it is usually the case that such networks are already path-consistent. Arc-consistency and path-consistency algorithms have time complexity of $\mathcal{O}(ed^2)$ and $\mathcal{O}(e^3d^3)$ respectively, where e is the number of variables and d is the size of the largest domain. Therefore, running consistency enforcement algorithms may be wasteful for many instances of FHSP if only constraint satisfaction is sufficient.

2.2.1 Row Convexity

A *globally consistent* network yields a backtrack-free solution, that is, a solution can be found in n steps where n is the number of variables in the network. As seen earlier, the constraints in the problem are fairly loose and usually path-consistent. Such networks can be identified to be globally consistent or *minimal* using the following theorem.

Definition 13 (Row Convex (Van Beek & Dechter 1995)) *A binary relation R_{ij} represented as a $(0,1)$-matrix is row convex if and only if in each row all of the ones are consecutive; that is no two ones in a row are separated by a zero.*

Theorem 2 ((Van Beek & Dechter 1995)) *If there exists an ordering of the domains $D_1, \ldots, D_n$ of a path-consistent network N such that all the relations are row convex, then N is minimal or globally consistent.*

Theorem 3 ((Booth & Lueker 1976)) *An $m \times n$ (0,1)-matrix specified by its f-nonzero entries can be tested for whether a permutation of the columns exist such that the matrix is row convex in $\mathcal{O}(m + n + f)$ steps.*

Furthermore, row convexity property can be checked in linear time as Theorem 3 suggests. Even though our network is path-consistent, the binary relations are not row-convex. This can be seen by analyzing the matrix representation of R_{ij}. Some rows in the middle of the matrix do not have the consecutive ones property unlike some rows at the top and at the bottom. Shifting the columns with non-consecutive ones so as to put all the ones consecutively, will always result in at least one other row containing non-consecutive ones.

2.3 Conclusion

We showed that a sub-problem of the CCSP can be viewed as a node-packing problem. The advantage of modelling the CCSP as a node-packing is the use of clique constraints which act as integral facets in the node-packing polytope. Finding integral facets for a particular integer programming problem has always been an interesting research issue. We believe that the clique constraints allow us to model the FHSP as a MIP using 0-1 integer variables and provide a good reason as to why this formulation of the adjacency constraint is better in practice.

We also showed that the constraint structure in the sub-problems of the CCSP is loose. That is, most practical consistency enforcement algorithms are not useful while solving the CCSP as a constraint network. Problem specific heuristic or iterative improvement may be more effective without the consistency enforcement.

Chapter 3

Solving the CCSP using MIP Models

The Forest Harvest Scheduling Problem (FHSP) and forest planning problems in general are combinatorial optimization problems. Briefly, the problem is to develop a forest management plan for a large area of forest satisfying multiple constraints and balancing competing objectives. Traditionally, mathematical programming has been and still is the most widely used technique for solving such problems. For the general forest harvesting problem, there exist a number of LP and MIP models (cf. literature review in Chapter 1). In this chapter we study a restricted version of the the FHSP, called the Clear-Cut Scheduling Problem (CCSP) (defined in Chapter 1.1) as a test case. We attempt to model and to solve the CCSP using a MIP model. We will not model the problem as a LP since the literature survey shows that LP-based strategies are difficult to interpret and may be impossible to implement, largely due to their inability to express spatial relationships.

We will start with an example of a MIP model for the CCSP from Johansen and Misund's paper (Johansen & Misund 1995). Their model comprises only the minimum height constraint (C_1), and the harvest interval constraint (C_2) as given in equation (1) and (2) (cf. Chapter 1.1). Their model is incomplete since it does not include any of the objective criteria. We will briefly describe this approach and outline its drawbacks. As a next step we present our MIP model formulation which deals with both constraints and objective criteria. We test our model using real-life data from the Norwegian ECOPLAN project (Misund *et al.* 1995). Our intention is to test the hypothesis that MIP models can only

be used effectively for small instances of the CCSP. Any practical instance of the FHSP is likely to be more complex and larger in orders of magnitude.

3.1 MIP Model I

Johansen and Misund's report describes their attempt to formulate the CCSP as a MIP model (Johansen & Misund 1995). They define n (real) variables, $h_1, \ldots, h_n$, where h_i corresponds to the harvesting period for stand S_i. They justify their use of real variables for harvesting periods by pointing out that if, for example, some h_i is 10.3, then this may give an additional information about when in the period 10 the stand S_i should be harvested. The report describes only the minimum height and the harvest interval constraints.

The harvest interval constraint is included in the model simply by permitting the values of h_i to be in the allowable harvest interval $[EARLY(S_i) + LH(S_i), LATE(S_i) + LH(S_i)]$, for all i. The interval may be relaxed if needed. Note that this does not reflect the definition of the optimal harvesting time objective (cf. equation (3) in Chapter 1).

The minimum height constraint is modelled by introducing a binary integer variable for each pair of neighboring stands (S_i, S_j). The value of the binary variable is either 0 or 1 and is determined by the following equation:

$$b_{ij} \leq 1 + \frac{h_i - h_j}{m+1}, \qquad b_{ij} \geq \frac{h_i - h_j}{m+1}$$

where m is the number of periods in the scheduling horizon. Now assuming that $h_i \neq h_j$ and MH be the time required for the stands to grow up to the minimum height for the harvest, the minimum height constraint can be represented by the following equations:

$$h_i - MH(S_j) - h_j > (b_{ij} - 1) * 2(m+1), \qquad h_j - MH(S_i) - h_i > -b_{ij} * 2(m+1)$$

The first equation is neglected if $b_{ij} = 0$, assuming that no stand takes more than the length of the plan to grow up to the height of X meters. If $b_{ij} = 1$, then it states that $h_i - MH(S_j) - h_j$ must be greater than 0. This reflects the X-meter constraint when $h_i > h_j$. The other equation works similarly.

Johansen and Misund's report (Johansen & Misund 1995) stops here as far as the MIP model is concerned. It mentions that the authors were unable to model other constraints and objective functions. Later in (Misund *et al.* 1995), the authors mention that this model is not suitable for the FHSP.

We tried to extend the above model by adding the rest of the objective criteria. We found it to be a difficult process. Even the minimum height constraint formulation seems awkward. Furthermore, we think that representing harvesting periods by real variables may make the solution hard to implement at the operational level. Therefore, we present another MIP model which is capable of expressing the objective criteria in addition to the spatial relationship constraints.

3.2 MIP Model II

We abandoned the real variables $h_i, \ldots, h_n$ and started with a binary (0,1) variable x_{ij} for each stand S_i, where $i = 1 \ldots n$ and $j = 1 \ldots m$, where n is the number of stands and m is the number of periods.

It may seem counter-intuitive to replace some compact-looking constraints with a large number of constraints with binary variables. But the reason it works well is that the CCSP, in this formulation, has node-packing as a subproblem (cf. Chapter 2). Therefore, we can expect to benefit from the clique constraints that are likely to quicken the MIP solution process. Furthermore, as stated in Chapter 2, this formulation has proven to work well in practice.

Our MIP model consists of n stands, $\{S_1 \ldots S_n\}$. Each harvest period p for a stand S_i is represented by a binary variable x_{ip}. We assume that each stand is harvested only once throughout the scheduling horizon. Now, we formulate the constraints and the objective functions for the CCSP as stated in the problem definition (cf. Chapter 1.1).

Constraints In addition to constraints C_1 and C_2, we have an additional constraint such that every stand is harvested exactly once during the scheduling horizon.

C_1: **Minimum height constraint** For all t from $1 \ldots m$,

$$x_{it} + \sum_{t-MH \leq n \leq t+MH} x_{jn} \leq 1, \ \forall \text{ neighbours } (S_i, S_j).$$

C_2: **Harvest interval constraint:** Simply do not include the binary variables associated with the undesirable periods for all stands in the model.

C_3: **At least one harvest per plan** $\sum_m x_{im} = 1, \ \forall i$.

Objective Criteria All except the objective criterion O_4 is formulated as follows.

O_1: **Optimal Harvesting Time** The actual time between harvests should be as close to the optimal as possible. For all i in $1 \ldots n$,

$$\sum_j x_{ij} * (|j - OPT(i)|) + \alpha_i \geq 0$$

where j goes from $1 \ldots m$, $\alpha_i \geq 0$. Here α_i is penalty for not harvesting stand i in its optimal harvest time. Let $P_{opt} = \sum_i \alpha_i$. Then, the optimal harvesting time objective is achieved by minimizing P_{opt} in the objective function.

O_2: **Even Harvest Volume** The estimated harvested volume $EHV_p(\mathcal{S})$ for any period p should be as close to average harvested volume in a schedule $\mathcal{S}$, $AHV(\mathcal{S})$. In other words, the variance between the harvest volumes should be minimized.

Let v_{ij} be estimated volume of forest obtained by cutting stand S_i in period j. Then, $V_j = \sum_{i=1}^{n} x_{ij}\ v_{ij}$, $\forall j$ is total volume harvested in period j. Set constraints

$$V_{min} \leq V_j \quad \text{and} \quad V_{max} \geq V_j, \quad \forall j = 1 \ldots n.$$

Then objective is to minimize $V_{max} - V_{min}$, which effectively minimizes the variance between volumes harvested, making an overall schedule with even consumption throughout the planning horizon.

O_3: **Old Forest** A specified minimum area of old forest (AOF), that is, stands with average age above a certain threshold, should be maintained throughout the scheduling horizon. Let $OLD(p, \mathcal{S})$ be the area of old forest in period p.

Let $O_{it} = 1$ if stand i is not cut before it becomes old in period t, otherwise $O_{it} = 0$. That is,

$$\sum_{1 \leq j \leq t-1} x_{ij} \ + O_{it} = 1\ , \quad \forall i.$$

Let θ_t be the area of old forest at period t, which is $\sum_i O_{it} * v_{it}$, $\forall i$. We want to keep θ_t as close to AOF (desired area of old forest) as possible. That is, for all t in $1 \ldots m$,

$$\theta_t - AOF + \gamma_t \geq 0, \quad \gamma_t \geq 0.$$

Then, the objective function will minimize $\sum_t \gamma_t$.

O_4:**Visual Impact** This is not put into the MIP model.

This type of model can be extended to handle alternative treatments. For an additional t treatments, $t(nm) - nm$ variables will be added to the model. The same order of constraints will also be added, of type $x_{ijk} + \sum_{k=2}^{k} x_{ijk} \leq 1$, which means that one treatment type inhibits other treatments for a certain number of periods. The assumption that only one treatment is allowed per scheduling horizon can also be dropped.

3.3 Adjacency Constraints Revisited

As noted earlier in Chapter 1, a significant amount of current research in operational forest planning is devoted to the formulation of adjacency constraints using integer variables in a MIP model. This constraint is the reason research shifted from LP to MIP in forest harvest planning and scheduling. Various formulations for this constraint exist in the literature.

The most straight forward and well-known method is known as the pairwise approach whereby every pair of adjacent stands, (S_1, S_2), share a constraint of type $x_1 + x_2 \leq 1$, where x_i is 1 if stand S_i is to be harvested and 0 otherwise. This approach was commonly used until Meneghin *et al.* proposed a new *type I* adjacency constraints (Meneghin *et al.* 1988). They proposed that only one constraint be shared between the mutually adjacent stands. Thus, if 2 stands are mutually adjacent, then they will share a pairwise adjacency constraint. But there is also a possibility of 3 or 4 stands being mutually adjacent. Then they will share a triplet or quatruplet constraints. These constraints altogether are called the type I constraints. The authors also present a method to identify a minimal set of these constraints. First, all quatruplet constraints are generated, then the triplets that are not contained in already identified quatruplet constraints are identified. Finally, all the pairwise constraints that are not proper subsets of a quatruplet or a triplet constraints are identified.

There are other methods proposed in the literature that identify minimal set of adjacency constraints. Examples include Ordinary Adjacency Matrix (OAM) constraint set (Yoshimoto & Brodie 1994), and *compartmental constraints* (Torres-Rojo & Brodie 1990, Murray & Church 1994) *et cetera.*

Church and Murray 95 review these formulations and give empirical evaluations on 6 different cases (Murray & Church 1995a). The authors conclude that the type I constraint formulation is the winner over all other formulations when solving these data sets in terms of computation time even though the number of constraints is higher than the others. However, they do not give an explanation for this. We believe that the explanation lies in the use

of the *clique inequalities*. The type I formulation allows the problem to be viewed as a set of node-packing subproblems. Recall the definition of a set-packing problem: Any problem that can be defined as a linear system of inequalities $Ax \leq b$, such that $\mathbf{A}$ is a $(0,1)$ matrix, $\mathbf{b}$ is an unit vector and $\mathbf{x}$ is binary. Any set packing problem can be transformed into a node-packing graph (Nemhouser & Wosley 1988). Using the type I formulation, A becomes a $(0,1)$-matrix, and b a unit vector (cf. 2). In other adjacency constraint formulations this is not the case. Therefore, these formulations may not generate integral facets. This may be why the type I constraint formulation came out as the winner in Church and Murray's empirical evaluation (Murray & Church 1995a).

Meneghin *et al*'s type I adjacency constraints can also be explained simply by using the graph-theoretic term clique. All mutually adjacent stands form a clique when each stand is treated as a node and adjacent relations as edges in a graph. Furthermore, such a graph will be a planar graph since it is a representation of a geographical region. It is a known result that a planar graph can not contain cliques of size 5 or more. Therefore, finding the clique inequalities for maximal cliques of size 4, 3,and 2 will be the minimal set of adjacency constraint formulation. Thus, what Meneghin *et al* found was precisely the clique inequalities.

3.4 Experimental Results

First we describe empirical results for random data, with only the adjacency and harvest interval constraints (C_1 and C_2), using a public domain MIP solver *lp_solve version 1.5* (*lp_solve* n.d.). Next, we present the results obtained from solving instances based on the ECOPLAN prototype data (Misund *et al.* 1995), using a commercial MIP solver *cplex version 4.0.4* (CPLEX n.d.) with all the constraints and the objective criteria as described in our MIP.

3.4.1 Initial experiments using *lp_solve* solver

Forest data was randomly generated using data description as described in (Misund *et al.* 1995). [1] This data is parsed and another input file is constructed for the MIP solver *lp_solve*. Only the adjacency and the harvest interval constraints (C_1 and C_2) were used during the data generation. The objective value was set so that the optimal value is n for

[1] This data set can not be compared with the data sets used with *cplex* solver

n	obj. value	remark
5	5	optimal
10	10	optimal
20	20	optimal
30	30	optimal
40	40	optimal
50	50	optimal
60 - 95	N/A	memory fault
100	100	optimal

Table 3.1: Results obtained for random data using the *lpsolve* MIP solver

instances with n stands. Our intention was to test how hard these problems were when considering only constraint satisfaction and if it was possible to solve the model with some additional objectives.

We generated problem instances for a variable number of stands, but the planning horizon was left constant at 100 periods with each period equivalent to 1 year. The number of stands were varied from 5 to 100. The results using a Sun sparc5 workstation is as shown in Table 3.1. The solution time varied from 5 minutes to 2 hours. The results show that for almost half the instances *lp_solve* exits abnormally with a memory fault. The rest are solved to optimality. For our randomly generated data, the neighboring stands for any given stand were low resulting in a loose adjacency constraint structure. This may explain why instances with 100 stands were solved to optimality. Another reason is the domain size of 100 periods. Although some of the periods were disallowed by the harvest interval constraint (C_2), every stand has still between 75 to 100 domain values to pick from. In fact, when the number of periods were reduced to 20, the solver was not very successful. Furthermore, memory faults caused by more than half the instances lead us to choose a more powerful MIP solver *cplex*.

3.4.2 Results using *cplex* solver

The Norwegian ECOPLAN project's prototype data was used to test our MIP model using *cplex* MIP solver. This data set is different than the one used with lp_solve solver in that the former has realistic adjacency relations and the number of periods for each stand. The area of each stand was generated randomly to occur between the largest and smallest area in the

actual data set since we were not able to get the data set with the real area information. All the instances were run on a Sun sparc10 workstation.

The ECOPLAN prototype data for 494 stands was divided into smaller instances of n stands where $n = \{25, 50, 100, 200, 300, 400, 494\}$. Each of these instances was tested with 5, 10, 15 and 20 periods (m) as the scheduling horizon. Furthermore, each of these instances was used to generate 5 different sets with different objective criteria, $S1, \ldots, S5$, as input for the *cplex* solver. Among these sets, $S1$ contains data for constraint satisfaction only. That is, only the adjacency constraint (C_1), the harvest interval constraint (C_2), and one harvest per scheduling horizon (C_3) were present (besides the integrality constraints) without any of the objective criteria. The objective value was set to be n for a data set with n stands. [2] All other sets also contain these constraints but the objective criteria differ in each of the sets. $S2$ contains objective O_1 (optimal harvest time), $S3$ contains objective O_2 (even harvest volume), $S4$ contains objective O_3 (old forest), and $S5$ contains all three objective criteria (O_1, O_2, and O_3).

Table 3.2 shows the cplex output for the ECOPLAN prototype data when the green-up age for adjacency constraint was set to 3 periods, with each period being equal to 5 years. [3] The table shows that it was difficult to satisfy most of the instances. There were no integer feasible solution for all of the data sets with 5 periods. Note that if an integer feasible solution does not exist for the set $S1$, then no integer solution exists for any other sets. Furthermore, if no integer feasible solution exists for a data set with n stands and m periods, no integer feasible solution exists for any of the other data sets with m periods and stands greater than n. [4] Only the data sets with pairs (25, 10), (25, 20), and (50, 10) were solved to optimality, where each pair represents (n, m).

The entries in the table marked with a "**" may seem awkward, for example, why is there no integer solution for the instance with 25 stands and 15 periods when there exists an optimal solution for the instance with 25 stands and 10 periods? The reason is that during the data generation, if the allowable harvest interval for a stand goes beyond the scheduling horizon, it is relaxed so that the stand may be allowed to be harvested during the last 5 periods. This was done so that every stand is harvested once. Thus, in the case of the instance with 10 periods, some stands got a larger harvest interval than in the case

[2] Amounts to cutting all stands once.

[3] This was the specification given to ECOPLAN project by the Norwegian Forest Owner's Association.

[4] Since all the adjacency relations are contained in the larger problem.

Set	n	m	read	C	V	opt	int sol	time	iter	BB nodes	remark
S1	25	5	.11	190	125	N	NA	.08	0	0	no int sol
S1	25	10	.20	296	250	Y	25	.18	42	5	
S2	25	10	.21	537	501	Y	103	.48	227	8	
S3	25	10	.12	326	262	Y	70035	18.27	6941	105	
S4	25	10	.20	573	532	Y	202177	2.69	1248	75	
S5	25	10	.18	844	795	Y	277051	27.93	8756	304	
**S1	25	15	.22	250	375	N	NA	.03	0	0	infeasible
S1	25	20	.21	369	500	Y	25	.14	17	42	
S2	25	20	.27	845	1001	Y	6	.28	30	9	
S3	25	20	.17	429	522	Y	315564	3.93	1904	43	
S4	25	20	.27	910	1041	Y	181096	9.72	2961	348	
S5	25	20	.36	1446	1564	Y	496684	8.54	2733	105	
S1	50	10	.21	766	500	Y	50	2.96	869	71	
S2	50	10	.26	1248	1001	Y	253	45.98	10974	1207	
S3	50	10	.25	796	512	Y	310699	4162.34	763278	22761	
S4	50	10	.34	1300	1043	Y	476260	76.74	18384	1214	
S5	50	10	.43	1812	1556	Y	843734	802.25	120204	3878	
**S1	50	20	.33	1084	1000	Y	NA	17.18	4683	1087	no int sol
S1	100	10	.31	1524	1000	N	NA	.15	0	0	infeasible

Table 3.2: Results obtained for Set 1 using *cplex* MIP solver

n: number of stands, m: number of periods, read: problem read time, C: number of constraints, V: number of variables,opt: is solution optimal?, int sol: integer solution, time: cpu time (in secs), iter: number of Branch and Bound (BB) iterations, BB nodes: number of BB nodes tried

with 15 periods.

Careful analyses of the solver output for the unsatisfiable instances revealed that the harvest interval constraint for some stands was too tight, that is, the stands could be harvested in only 1 or 2 periods. This caused the solver to conclude integer infeasibility relatively quickly. We were interested in testing the MIP solver for larger instances with a number of objective criteria. Therefore, we relaxed the harvest interval constraint so that a stand can be harvested in any period during the scheduling horizon. This is not an unrealistic relaxation since the minimization of the objective O_1 essentially has the equivalent effect on the solution. Also, to increase the number of periods, we only generated data with 20 periods.

	Set 2						Set 4					
n	C	V	O_1	time	CC	nodes	C	V	O_3	time	CC	nodes
25	610	500	*6	2	13	2	1151	1041	*180	46	36	87
50	1740	1000	*45	17	64	54	2781	2041	*0	350	47	102
75	1926	1500	*83	37	67	57	4251	3041	*0	380	116	112
100	3640	2000	*115	53	87	37	5681	4041	*0	202	22	120
150	5540	3000	*203	130	123	91	8581	6041	NA	7200	115	75
200	11482	8001	*307	2538	245	3776	1161	8041	NA	7200	101	144
250	14781	10001	367	7200	303	9090	15021	10041	*0	2016	297	189
300	18264	12001	468	7200	291	11488	18541	12041	NA	7200	123	151
350	220041	14001	527	7200	426	5033	22361	14041	NA	7200	382	97
400	25081	16001	601	7200	420	5014	25441	16041	NA	7200	179	56
450	28264	18001	669	7200	INT							

Table 3.3: Results obtained for Set 2 and 4 using *cplex* MIP solver

n: number of stands, C: number of constraints, V: number of variables, O_1: optimal harvest time value (* $\equiv$ opt), O_3: old forest objective value (* $\equiv$ opt), time: cpu time (secs), CC: number of clique cuts applied, nodes: number of Branch and Bound nodes tried, INT $\equiv$ inturrupted

Data instances with $n = \{75, 100, 150, 200, 250, 300, 350, 400, 250, 494\}$ was generated for all 5 sets $S1$ to $S5$. While solving some instances it was observed that the solver was taking more than 2 hours just to find an initial solution. These instances were rerun with the solver's rounding heuristic turned on. The solver used a sophisticated rounding heuristic to find the first integer solution. In addition, the solver had also clique and cover cuts options to improve performance of the branch and bound phase.

In Table 3.3 smaller instances have been solved to optimality. The objective value for O_1 denotes the total number of periods off the optimal harvesting period for all the stands. For example, if two stands were harvested 5 periods either before or after the optimal, and the rest were scheduled on the optimal harvesting time, then the value of O_1 is 10. The value of the objective O_3 (in thousands of square kilometers) is the total area of forest not above the old forest threshold for all the periods. The old forest threshold was set to 10% of the total area for all the periods in our experiments, which comes to 1440 square kilometers (in thousands). The solver could not find optimal solutions for the set $S2$ starting from $n = 250$. For the set $S4$, no integer solution was found within the time limit of 2 hours for n starting from 150.

	Set 3					Set 5						
n	C	V	O_3	CC	nodes	C	V	O_1	O_2	O_3	CC	nodes
25	670	522	114.8	219	36213	1687	1564	125	73.8	180	144	9765
50	1800	1022	361.4	541	6387	3795	3064	221	323	*0	272	4907
75	2770	1522	552.2	480	1553	5746	4564	209	504	*0	280	1326
100	3700	2022	317.6	610	394	7655	6046	355	452	*0	284	492
150	5600	3022	366	658	933	11516	9064	511	556	*0	377	525
200	7700	4022	455.2	382	506	15583	12064	no int	NA	NA	165	190
250	10040	5022	202.6	502	514	19882	15064	no int	NA	NA	215	150
300	12560	6022	no int	269	93	24365	18064	no int	NA	NA	287	236
350	15380	7022	no int	332	30	29142	21064	no int	NA	NA	359	197
400	17460	8022	no int	228	61	33182	24064	no int	NA	NA	389	4
450	19680	9022	INT									

Table 3.4: Results obtained for Set 3 and 5 using *cplex* MIP solver

n: number of stands, C: number of constraints, V: number of variables, O_1: optimal harvest time value, O_2: even consumption objective value, O_3: old forest objective value (* $\equiv$ opt), CC: number of clique cuts applied, nodes: number of Branch and Bound nodes tried, INT $\equiv$ interrupted

Table 3.4 shows results for the sets $S3$ and $S5$. The value of the objective O_2 (in thousands of square kilometers) is the difference between the biggest and the smallest harvest volume during the scheduling horizon. All the instances were stopped after 2 hours of execution, that is, no optimal solution was found. No integer solution for the set $S3$ was found for values of n starting from 300. For the set $S5$, this value was 200.

In all the instances, the number of clique cuts generated was high. This means that turning this option on during solving was probably a good idea since these cuts give tighter bounds and help branch and bound algorithm to find an integer optimal solution faster.

After the elimination of the harvest interval constraint, the solver was able to find the optimal solution for all of the instances in $S1$, that is, the adjacency constraint was satisfied. When the objective criteria were added, the solver required more time to find the optimal or an integer solution. From the results, it seems that the addition of the even harvest volume objective makes the MIP much harder since both the sets $S3$ and $S5$ that contain the objective were not solved to optimality even for smaller instances. It remains to be seen how good these solutions are since currently we have no means of comparing them. One measure could be the optimal solution of the full problem with $n = 454$. However, running

the problem for 33 hours did not even produce an integer feasible solution.

These results show that the solver is capable of finding integer solutions to the ECOPLAN prototype data when the harvest interval constraint is relaxed and no objective criteria is in the model. Also, an optimal solution was found for smaller instances (usually 25 and 50) even after the addition of all of the objective criteria. However, the rest of the instances were either not solved to optimality or not even to an integer feasibility. This suggests that for practical problems, with orders of magnitude larger than those of our test problem, this method alone is not realistic.

3.5 Integrating LP with Simulation

As noted earlier in Chapter 1, the growth and treatment simulation are an important part of the forest harvest scheduling procedure and our goal is to integrate the two processes. In this section we will describe Hoen's work on integrating the simulation with a LP solver (Hoen 1992) . The simulator used is the growth and treatment simulator called GAYA and the integrated product is called GAYA-LP.

If N is the total number of schedules and K the number of stands, Hoen formulates the LP problem of maximizing net present value (NPV) of the harvest schedule as follows:

$$\max \quad \mathbf{Z}_p \quad = \quad \mathbf{c}^T\mathbf{x}$$

subject to

$$\mathbf{A}_1\mathbf{x} \leq \mathbf{b}_1; \quad \mathbf{A}_2\mathbf{x} \leq \mathbf{b}_2; \quad \mathbf{x} \geq 0$$

where

$\mathbf{Z}_p$ The objective function. Net present value (NPV) is most commonly used.

$\mathbf{x}$ N $\times$ 1 (column) vector. Decision variables representing the number of hectares to be treated by a management schedule (activity).

$\mathbf{c}^T$ 1 $\times$ N (row) vector containing NPV value of schedule.

$\mathbf{A}_1$ M $\times$ N matrix produced by the simulator. It contains production coefficients, net payment, and harvest volumes for each schedule.

$\mathbf{b}_1$ M $\times$ 1 (column) vector of resource constraints.

$\mathbf{A}_2$ K × N matrix of area related coefficients, where element (ij) is 1 if activity j belongs to stand i, otherwise it is 0.

$\mathbf{b}_2$ K × 1 (column) vector consisting of the initial area of the stands.

The problematic part about the simulation is that it produces a large number of treatment schedules for the stands in a forest, often in the order of thousands. This results in a LP formulation with a large number of columns as compared to the rows. We can see this in the above LP formulation as well, where the number of columns in the LP matrix is the total number of schedules for all of the stands and the number of rows are the resource and the area constraints. Column generation as a problem decomposition technique can be used to increase computational efficiency of this model (for detail see (Chvatal 1983)). The solution of the LP problem can be determined incrementally by looking at a subset of the column, n_a. The problem with n_a columns, LP_a, is solved. The dual variables from the solution of LP_a is used to evaluate an entering variable among $N - n_a$ columns for the basis of the primal LP_a problem. The reduced cost for $N - n_a$ column is calculated and columns with positive costs are chosen as candidates for the entering variables. Then, non-basic variables from optimal solution of LP_s are replaced with the candidate variables and the problem is resolved. This process continues until the optimal solution to the original LP has been found or certain other criteria are met.

Hoen reported solving models with only a few hundred constraints using GAYA-LP. Thus, it has not proven to work at all with a large number of stands and schedules as would be the case in any practical problem. Furthermore, the disadvantage of using this model is that no constraints between the stands are represented. Therefore, the plans generated by GAYA-LP may not be feasible spatially. That is, taking x hectares with management activity a, as suggested in the plan, may not be possible. Therefore, adjacency constraint has to be satisfied in the solution. But, the addition of this constraint in the model may not be a trivial task. Also, it is not obvious how to model the objective criteria like even consumption, old forest *et cetera*.

GAYA-LP is an example of loose integration of the simulation with an optimization process. All possible schedules are compiled in advance before running the LP solver. If the LP solver can ask the simulator to generate a number of additional schedules as it progresses, then we can achieve the true integration of a simulator and an optimizer in the harvest scheduling process. No LP or MIP scheduling system for forest harvesting to date

(to our knowledge) has achieved this level of integration.

3.6 Conclusion

From our experimental results we can conclude that our MIP model can be used for the CCSP and that small instances are solved to optimality if only some objective criteria are used. For larger instances the harvest interval constraint was too tight and had thus to be relaxed. Then, satisfying the adjacency constraint was easy.

It was when we started introducing some additional objectives, that the solving started to become time consuming. In particular, the even flow objective seemed to be hard to optimize. The clique and cover cuts option in the solver was turned on while solving these instances. A high number of clique cuts was generated for all the instances suggesting tighter bounds for the branch and bound phase. However, the largest instance with all the objective criteria was not even solved for an integer feasible solution after 33 hours of execution in a Sun sparc10 workstation. Furthermore, *cplex* is considered to be the state-of-the-art among the current MIP solvers. The MIP can be made more efficient if the problem decomposition technique called column generation can be utilized. However, modelling the constraints and the objectives to use this technique is not a trivial task. Thus, our MIP model alone is not realistic for solving large practical FHSP.

Chapter 4

Solving the FHSP using a CSP model

The Forest Harvest Scheduling Problem (FHSP) is a complex combinatorial optimization problem with multiple constraints and objectives. The most popular method to solve these problems is based upon mathematical programming. However, as seen in Chapter 3, the MIP model for FHSP can be solved only for the small instances of the problem. Usually, the size of the model for large problems is too big for current MIP solvers. After testing the MIP model, we believe that unless problem specific heuristics are developed and used along with the MIP solver, it is very difficult to find solutions to the FHSP that is readily acceptable in a reasonable time. Since the real data is of an order of magnitude larger than the ones we have solved, this method alone is not sufficient. Hence, it is most likely that the large FHSP will require some other technique. A suitable candidate is Iterative Improvement Technique (IIT).

Iterative improvement methods can be described in general as consisting of 3 algorithms - A, B and C (Papadimitriou *et al.* 1990, Johnson *et al.* 1988). The algorithm A finds an initial solution. The algorithm B checks for the optimality of the solution and the algorithm C finds a new feasible solution. The process is iterated on algorithm B and C until a terminal condition is met. The process can be interrupted any time and the most recent - the best so far - solution can be obtained.

For the overall success of such algorithms in practice, it is believed that algorithm A has to give a "good" initial solution, not just any feasible solution (Papadimitriou *et al.*

1990). Since we are aiming at solving a large forest harvesting problem, it is worthwhile to spend some time in generating a good initial solution. The measure for this goodness is the number of consistent variables.

In this chapter, we outline a few approaches to generate a good initial solution for the FHSP. Next, we choose a method based upon Constraint Satisfaction Problem (CSP) model. This initial solution can be used by an iterative improvement technique for further optimization. This method incorporates forestry knowledge in the problem solution process by using the growth and treatment simulation. In the process, we develop a C++ class library called FHSPlib.

4.1 Methods Studied for Generating Initial Schedule

Different approaches can be used to generate an initial solution for the FHSP depending upon the model used to describe the FHSP. The most popular models are based on mathematical programming where one formulates the problem as a LP or a MIP. Then, the respective solvers look for the optimal solution of the model. An alternative model is based on the CSP which allows one to use the constraint programming techniques, such as constructive or iterative search methods, to look for the solution. We give a brief description of how these approaches may be used to generate an initial solution for the FHSP. We assume that the growth and treatment simulator (for example, GAYA (Hoen 1992)) is to be included in the solution process so that more realistic solutions can be obtained.

4.1.1 Linear Programming

In Hoen's GAYA-LP, a linear programming solver linked with the simulator GAYA, GAYA generates all possible schedules for all of the stands and then the LP solver selects one eschedule for each stand, depending upon an economic objective NPV (Net Present Value) of the forest. Hoen has integrated JLP (Lappi 1992), a specialized LP solver, with GAYA, and this reportedly works better than GAYA-LP [1].

There are three approaches based on GAYA and JLP. We will say that GAYA outputs a tree for each stand. Each path in the tree is a schedule. The number of leaf nodes in the

[1] Siitonen's MELA (Siitonen 1993) and Baskent and Jordan's GIS-based method (Baskent & Jordan 1991) are alternatives to GAYA.

tree is the number of possible schedules. Figure 1.2 in Chapter 1 is a small example of the tree.

1. Make singleton schedule for each stand, so that GAYA will only produce one path for each stand. This could be treated as a simple initial solution. The path chosen by the simulation for each stand depends upon the overall quality of the stand and various other forestry criteria.

2. According to Hoen, GAYA can be modified such that only a few schedules are generated for each stand based upon the economic criteria. If this is the case, then we should be able to generate any number of schedules per stand as required. Observe that this approach is not independent – it requires an additional LP solver.

3. GAYA-JLP can be used to select the optimal schedule for each stand based upon the economic criteria.

The first approach above can be implemented as a function that takes as input an individual stand data. The second approach seems like a good idea, provided that we can control the number of schedules generated per stand. Otherwise, there can be size problems with GAYA. The number of paths generated as possible schedules for each stand is quite large. For 10 time periods and 50 different treatments, it could be as large as 1000 (per stand). Only a small subset of the treatment set is possible depending upon the current state of a stand. All of these treatments finally translate into less than 10 basic treatments such as clear-cut, commercial thinning, and fertilization. In one case, GAYA-JLP was used for 6000 hectares of forest, approximately 14,000 – 15,000 stands, and 7 ten-year periods. For this problem, there was an average of about 25 schedules per stand generated by GAYA.

Use of JLP (in the third approach) requires definition of an objective function. The NPV seems to be the most commonly used for this purpose. Additional data - the number of trees per stand and the distance the trees have to be transported after being harvested - is required to calculate the NPV. The ECOPLAN prototype data that we used for our experiments in Chapter 3 do not have this information.

Summarizing, we need

1. GAYA formatted input data

2. Economic data to calculate the NPV for use with GAYA and with the LP solver part

Given this, a restricted (relaxed) version the FHSP may be solved to obtain the optimal solution which could be used as an initial solution for the full FHSP.

4.1.2 Mixed Integer Programming

The FHSP can be better modelled as a MIP. But the problematic part of this approach is the size of the model, since the number of variables and constraints become prohibitive even for a relatively small problem. Therefore, before using this model, a decomposition of the problem is required. This can be done by either partitioning the problem into smaller subproblems or using the MIP formulation that allows a column generation technique. To our knowledge there is not any system that has integrated the simulation model into a MIP model.

If the problem is geographically partitioned into subproblems, then the subproblems can be solved independently and the solutions combined. If it is the case that FHSP has constraints that are fairly loose from one geographical area to another, it may be possible to combine the subproblem solutions into a global one without much effort. However, the problems with this approach are that it is not obvious how the simulation results can be used and that current MIP solvers can only handle small problem instances. In short, making this approach work efficiently is not a trivial task.

4.1.3 The CSP model and Constraint Programming

Let $X = \{x_1, \ldots, x_n\}$ be the stands in a given forest, $D_i = \{P_1, \ldots, P_k\}$ be the set of paths obtained from a simulation for stand x_i, and $R_{ij} \subseteq D_i \times D_j$. Each path $P_i = \{a_1, \ldots, a_m\}$, where each a_i is a node in the path consisting of a tuple $< p_i, t_i >$, where t_i is a treatment and p_i the treatment period. We can use the adjacency constraint to construct $C = \{c_{ij}\}$, where c_{ij} is a set of allowable pairs of values for x_i and x_j. Then, by definition, the triple (X, D, C) is a CSP, where X is a set of variables, D is a set of domains and C is a set of constraints. If we draw a constraint graph for this CSP, each node in the graph represents a stand and each edge an adjacency constraint between two nodes (see Figure 4.1). This will give us a binary constraint network (cf. definition in Chapter 2). Let us assume that we want to use the results of the simulation [2] in this model.

[2] for example, GAYA

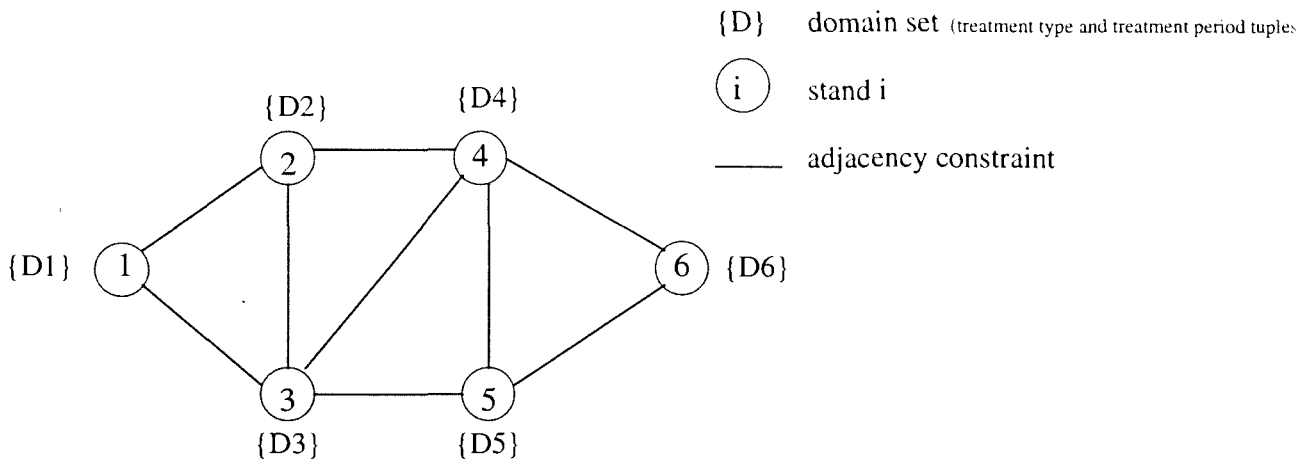

Figure 4.1: Constraint graph of a forest in Figure 1.1

1. start instantiating the variables in some static ordering $x_1, \ldots, x_n$. Then, $x_1 \leftarrow P_i \in D_1$.

2. for $i = 2$ to n, $x_i \leftarrow d_j \in D_i$ such that adjacency constraints are satisfied greedily with $x_1 \ldots x_{i-1}$.

3. if not all consistent, use local repair heuristic, for example, *minconflicts* (Minton *et al.* 1992), or tree search to generate an initial feasible solution

It is obviously not possible to represent all of the possible paths from the simulation as domains. A smaller subset of the set of paths have to be used for efficiency. We can then order the variables and start instantiating (step 1 from above). But to take care of the adjacency constraint, each schedule should have the *exclusion period* information, that is, the periods at which the stand is below the minimum height. This can be obtained from the input data directly using the age and growth information.

Instead of using a repair method in step 3, we could use a tree search. However, we believe that the problem has exponential search space and that therefore the tree search may not be a good idea.

Note that the ordering does not have to be in terms of the stands but can be in the periods. One can start instantiation from the period 1 to the period m. The algorithm moves from period i to period $i+1$ when a local objective criterion, for example, the area or the volume of the harvest, is fulfilled. We can check during each period to see if the assignments satisfy the adjacency constraints with the assignments from the previous periods.

Another approach could be to randomly select the treatments and the periods from the simulation result. But the solution in most cases may not be feasible since the simulation result does not take into account any constraints between the stands including adjacency constraints. However, the solution can be made feasible by using an iterative repair heuristic or a constructive tree search method.

Summarizing, to use a CSP model we will need

1. a small number of paths generated for each stand by a simulator (item 2 from previous section)

2. a suitable constructive search or an iterative repair method

4.1.4 Conclusion

It seems that in order to use a MIP model to generate an initial solution, various other problems like decomposition, suitable constraint formulation *et cetera* have to be considered first. Linear programming model with the GAYA simulator seems to be an easier choice. Even GAYA alone could be used to generate a simple initial solution if only one schedule is picked for every stand. A CSP model is easier to formulate and a number of solution strategies can be adopted from constraint programming. The approaches that use simulation results intuitively seem to be better since the growth and yield models are then incorporated in the solution process. We give a method based on a CSP model in the next section.

4.2 A method based on a CSP model

The FHSP problem is modelled as a CSP and a well known *minconflict* heuristic is used to iteratively repair a greedily assigned initial (trial) solution. We will describe our implementation and empirical evaluation of this method in the following sections. As noted earlier, we develop a C++ class library, FHSPlib, in this process. The implementation uses

an external C++ class library called LEDA [3].

In this section we will focus more on the input requirements and on the outputs from the FHSPlib library. A typical application while using this library would also have a data generator which prepares the data for the problem instantiation routine in the library. The data generator typically works on the simulator results to prepare a number of schedules for the stands.

The input data file is assumed to have a particular format, which is the output format of the data generator (or data preprocessor). Appendix A has a detailed explanation of the format.

Some of the data items (for example, the number of trees, the volume *et cetera*) are not currently used in the initial solution generation process but they are kept there assuming that all the results from the stand simulator will be used in the future.

4.2.1 Minconflicts Heuristic

It is often the case that while solving a CSP using the iterative techniques, an initial trial solution is generated randomly or greedily. In most cases this solution is infeasible as it violates various constraints. One way to make this a feasible solution is iteratively repairing it by changing the value assigned to a variable. Which variable to choose for the change is a research issue and often depends upon the problem. One of the methods of choosing the variable to change is the minconflicts heuristic. This heuristic is very simple-minded, yet it has proven to be effective on large scheduling problems. An algorithm using it may be described as follows:

Algorithm 1

1. create initial trial solution
2. pick a variable, X, in conflict
3. assign value $x_i \in domain(X)$ to X such that the conflicts created by it is minimal
4. if no variables in conflict exit
5. else goto step 2

[3] Library of efficient data structures and algorithms (http://www.mpi-sb.mpg.de/LEDA/www/)

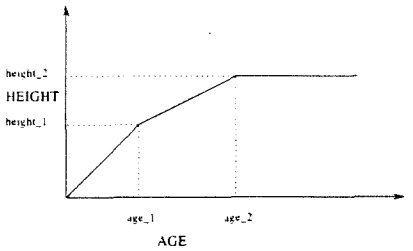

Figure 4.2: Function used to calculate height

Step 1 of the algorithm can be implemented either greedily or randomly. We have chosen a greedy way in our implementation. This step also creates two sets of variables, $VarsDone$ and $VarsLeft$ - with the former containing the consistent and the latter containing the inconsistent variables. Step 2 picks a variable - we pick randomly - from the set $VarsDone$. In step 3, all the values for the variable picked are checked and the one that minimizes the number of conflicts with other variables is assigned to it. Then, the two sets are updated. Step 4 checks for the terminating condition. In our implementation the number of iterations is also included as the terminating condition.

4.3 Experimental Results

We have a data generator that takes the following input parameters and produces stand specific data suitable for the library routines. The data file used (first argument) was the ECOPLAN prototype data (Misund *et al.* 1995). The source code for the data generator and the test application are found in Appendix B and Appendix C respectively.

For the height calculation, a simple linear function as shown in Figure 4.2 is used. The necessary points in the function are given as input parameters to the data generator. A more detailed parameter information is shown below.

```
The executable gen-data takes 10 arguments,

char* file = argv[1];              // data file
int MIN_AGE = atoi (argv[2]);      // for clearcut
```

```
int MAX_AGE = atoi (argv[3]);      // for clearcut
int age_1 = atoi (argv[4]);        // for height calc
int age_2 = atoi (argv[5]);        // ...
float height_1 = atof (argv[6]);   // ...
float height_2 = atof (argv[7]);   // ...
int nSch = atoi (argv[8]);         // number of schedules (domain size)
int period = atoi (argv[9]);       // number of periods
int pLength = atoi (argv[10]);     // period length

Example:
gen-data stands.db 50 80 20 110 18.0 22.0 1 10 5
```

4.3.1 Data Sets

Two basic data sets were used, both generated using the ECOPLAN prototype data. However, these sets are not comparable to the sets used with cplex MIP solver in Chapter 3. The latter donot have treatments other than the clear-cut and are restricted to one treatment per stand. These restrictions were primarily placed due to the inefficiency of general integer programming algorithms. The two data sets generated for this experiment with minconflicts heuristic have multiple treatments and a stand may have various treatments during the scheduling horizon. The difference between the parameters used to generate these two data sets was the values of parameters MIN_AGE and MAX_AGE, which determined the legal interval (in years) for the clear-cut treatment. For $DataSet_1$, and $Dataset_2$, the intervals were [35, 90] and [30, 100] respectively. From each set, further test sets were generated using the following argument to the data generator.

```
castberg-stands.hp 35 90 55 150 10.0 22.0 $nSch$ $nPer$ 5 ,
```

where $nPer$ is set $\{10, 15, 20\}$ and for $nPer = 10$, $nPer = 15$, and $nPer = 20$, respective values for $nSch$ were sets $\{5, 10\}$, $\{5, 10, 15\}$, and $\{5, 10, 20\}$. Thus, a total of nine test sets were generated from each data set.

$Datasetet_1$ was tested with 10, 100, and 500 iterations. At this point, it was observed that some of the consistent assignments were the schedules that had no treatment for all of the periods. This was because of the legal treatment interval which prohibited some

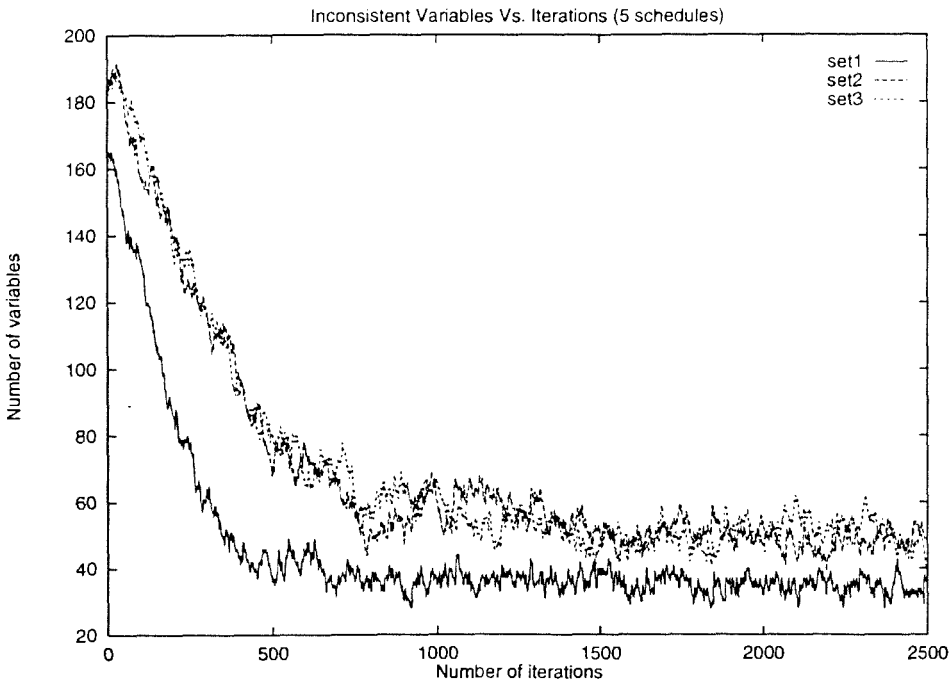

Figure 4.3: Variables left inconsistent and iterations (each stand has 5 schedules)

stands from having treatments.[4] Since this should not happen when using a simulator, the data generator was modified such that all the schedules generated had at least one valid treatment. This was done by randomly picking one period in which to allow the treatment. $Dataset_2$ was generated in such a way and used for further experiments. For clarity, we divided $Dataset_2$ into 3 sets, Set_1, Set_2, and Set_3, with $nPer = 10$, $nPer = 15$, and $nPer = 20$ with a respective $nSch$ of sets $\{5, 10\}$, $\{5, 10, 15\}$, and $\{5, 10, 20\}$.

4.3.2 Results

Figure 4.3 shows the result of running our algorithm for instances in which each stand has 5 schedules in its domain, from each of the three sets. Figure 4.4 is a similar plot but with instances where each stand has 10 schedules in its domain. In Figure 4.5, results from instances of Set_2 and Set_3 only are plotted, with each stand having 15 or 20 (only in Set_3) schedules as domains.

The figures show that after some iterations the number of inconsistent variables starts

[4] For example, there are some stands with current age greater than 100.

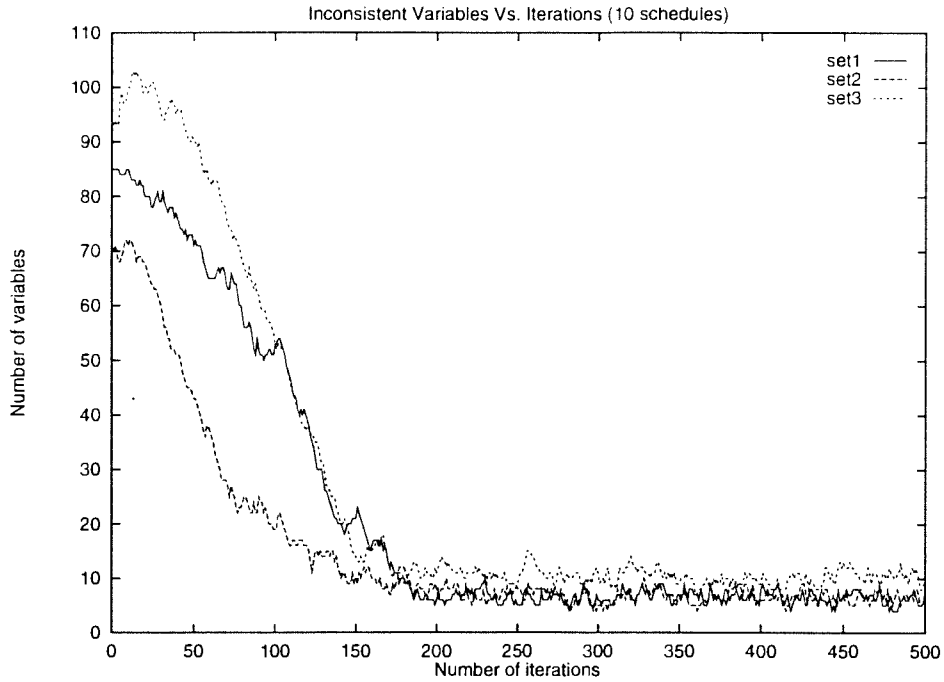

Figure 4.4: Variables left inconsistent and iterations (each stand has 10 schedules)

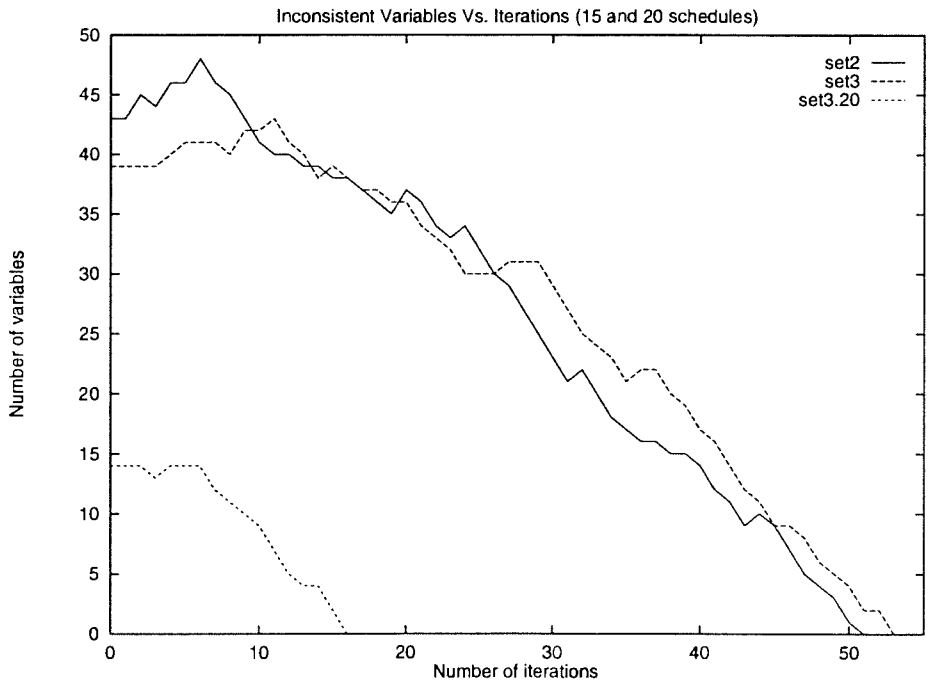

Figure 4.5: Variables left inconsistent and iterations (each stand has 15 or 20 schedules)

to fluctuate within a small interval. The position of this interval depends upon the number of schedules initially available to each stands. In Figure 4.3 and Figure 4.4, this interval is [30,60] and [5,10] respectively. In Figure 4.5, all the variables are consistent within 60 iterations. In fact, this interval is the largest for instances with the least number of schedules as domain. This is as expected - the bigger the size of the domain, greater the chances of finding a consistent solution with less numbers of iterations.

Furthermore, all the instances in Figure 4.3 and Figure 4.4 were allowed to run till 10,000 iterations in hope of finding a complete consistent instantiation. However, the number of the inconsistent variables remained within a constant interval throughout the run. Either, our algorithm was trapped in the local minima or there were no better solution to be found.

The results show that this method can be used to find the initial solutions. It also suggests that for the ECOPLAN prototype data, if 10 or more schedules are generated for each stand using a simulator, then the chances are that the method will converge to a complete or almost complete consistent instantiation. This is not a very demanding requirement since a stand treatment simulator like GAYA (Hoen 1992) can be modified easily to output such schedules. However, it is still to be seen whether the schedules generated by such a simulator behave as the randomly generated ones in our experiment.

The repair algorithm took 5–7 minutes of processing time in a sparc-10 SUN workstation for each of the instances.

4.4 Further Improvements and Suggestions

There are many ways to improve the performance of the FHSPlib library. The *minconflicts* heuristic implementation requires a programmer to make certain choices while implementing the trial solution generation process, the conflict set builder *et cetera*. The following approaches may improve the performance of the library.

1. Right now the graph class from LEDA is minimally used. However, it can be utilized to order the variable set employing some other heuristic. The rational for not using it now is that it creates an overhead that is not necessary in our test case. If the test case is complicated and large, then it may be beneficial to cluster the nodes of the graph according to their degree, neighbors, topology (for example, cliques), *et cetera* to order the variables before starting the repair algorithm.

2. The initial trial solution can be constructed randomly instead of greedily if the number of the solutions is large. The greedy assignments add overhead that may not be necessary.

3. It might be useful to introduce some optimization, such as the area harvested, the volume harvested, the net present value, *et cetera*, in addition to constraint satisfaction.

4.5 Conclusion

We implemented a library that provides users to represent the FHSP like problems as a CSP. We mostly focussed on the initial schedule building mechanism for the FHSP. The experimental results showed that our test case can be easily solved if only the adjacency constraint is considered. Other constraints are usually formulated as objectives which can be iteratively optimized. It also showed that the number of schedules available for each stand directly effects the running time and the quality of the initial solution. The quality of the initial solution also depends upon the input data provided. Thus, if the stand simulator is robust and can provide good varying schedules for each stand, then the initial solution builder can also provide a good starting point for iteratively optimizing the solution.

Overall, this method has shown to be useful for our test data which is a real-life prototype data from Norway. However, more empirical evaluations with larger data sets and simulator-generated schedules have to be done to fully evaluate the usefulness of this approach.

Chapter 5

Conclusion

Forest harvest scheduling is a multi-criteria optimization problem that requires multi-disciplinary expertise. Most of the research to date has been in mathematical programming based models for solving the problem. We studied the MIP and the CSP as both modelling and the solution strategies for this problem. Our study of the adjacency constraint structure has shown that some of the formulations of the constraint help the MIP solution process by giving tight bounds to the problem. Furthermore, the adjacency constraint seemed to be "loose", that is, consistency enforcement algorithms, like arc-consistency, may not be worthwhile in these cases.

We gave a complete MIP model for a restricted version of the problem called the CCSP. We tested the model with a real-life data from the ECOPLAN project in Norway. The results showed that the model is efficient for small instances of the CCSP and that the optimal solution can be found very quickly. However, we were not able to solve the full problem to optimality even after running the solver for 34 hours. This test data is very small as compared to the practical problems that exist. Furthermore, the growth and treatment simulation results were hard to integrate into the MIP model. Therefore we gave an alternative model based on CSP and showed how the simulation can be integrated into it.

We implemented a method that provides the users with a way to represent the FHSP like problems in a CSP setting. In particular we focussed on the algorithms for building the initial solutions using the growth and treatment simulation results. Our test results showed that the ECOPLAN prototype problem can be easily solved if only the adjacency constraints are considered. Other constraints are usually formulated as objectives which can

be iteratively optimized. Tabu search and simulated annealing are good candidates for the iterative improvement technique. We also showed that the number of schedules available for each stand directly effects the running time and the quality of the initial solution. The quality is also dependent upon the input data provided. The generated initial solution can be improved by fully integrating the stand simulator combined with minconflicts heuristics.

The FHSP problem has a significant spatial component, consider for example, the adjacency constraints. Recent advances in geographical information technology (GIT) should be exploited to handle, analyze and visualize the data. This is particularly the case when evaluating the visual impact of a schedule, and in general while handling ecological, recreational and esthetic constraints/criteria. We believe that the CSP model can be easily integrated with the GIT.

We did not solve the FHSP in this thesis but rather provided a good foundation for both modelling and solving the problem using either the MIP or the CSP technique. Future research would be to formulate the constraints and the objectives in the MIP model so that the column generation technique can be utilized. On the CSP area, the initial solution builder should be extended to solve the complete problem. Also, both of these approaches should be compared using the same data and penalty functions for all of the objectives. We believe that this will help advance the state-of-the-art in the forest harvest scheduling practices.

Bibliography

E.Z. Baskent & G.A. Jordan, 1991. Spatial wood supply simulation modelling. *The Forestry chronicle*, 67-6 (1991), 610–621.

P. van Beek, 1994. On the Inherent Level of Local Consistency in Constraint Networks. *AAAI-94 Conference, Seattle*, (1994).

P. van Beek & R. Dechter, 1995. On the Minimality and Global Consistency of Row-Convex Constraint Networks. *ACM Journal of Computing*, (1995).

J. R. Bitner & E. Reingold, 1975. Backtrack programming techniques. *Communications of ACM*, 18-11 (1975), 651–656.

K. S. Booth & G. S. Lueker, 1976. Testing for consecutive ones property, interval graphs and graph planarity using p-q-tree algorithms. *Journal of Comp. System Science*, 13 (1976), 355–379.

V. Chvatal, 1983. *Linear Programming.* W. H. Freeman and Company.

W. Conover, 1980. *Practical nonparametric statistics,* 2^{nd} *ed.* New York, Wiley.

CPLEX. Version 4.0.1, commercial MIP solver from CPLEX Optimization Inc. http://www.cplex.com/index.html.

L.O. Eriksson, 1994. Two methods for solving stand management problems based on a single tree model. *Forest science*, 40-4 (1994), 732–758.

E. C. Freuder, Nov 1978. Synthesizing Constraint Expressions. *Communications of ACM*, 21 (1978), 958–966.

E. C. Freuder, 1982. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29 (1982), 24–32.

O. Garcia, 1990. Linear Programming and related approaches in forest planning. *New Zealand Journal of Forestry Science*, 20 (1990), 307–331.

C. Gebotys & M. Elmasry, September 1993. Global Optimization Approach for Architectural Synthesis. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 12-9 (1993), 1266–1278.

F. Glover, 1989. Tabu Search - Part I. *ORSA Journal on Computing*, 1:3 (1989).

H. F. Hoen, 1992. GAYA-LP: A PC-Based long range forest management model. In *EURO XII/TIMS XXXI Joint Intl Conference*, Helsinki, Finland.

J. Hof & L. Joyce, 1992. Spatial optimization for wildlife and timber in managed forest ecosystems. *Forest Science*, 38 (1992), 489–508.

K. Hoffman & M. Padberg, Spring 1991. Improving LP Representations of 0-1 Linear Programs for Branch-and Cut. *ORSA Journal on Computing*, 3-2 (1991), 121–134.

B. S. Johansen & G. Misund, SINTEF Internal Report on GISHP 1995. The Forest Harvesting Problem.

D. S. Johnson, C. H. Papadimitriou, & M. Yannakakis, 1988. How easy is local search. *J. Computer and System Science*, 37 (1988), 79–100.

K. N. Johnson & S. Crim, 1986. FORPLAN version I: Structure and options guide. Technical report, USDA Forest Service Land Management Planning System Section, Washington, DC.

K. N. Johnson & D. B. Jones, 1979. MUSYC user's guide and operation manual. Technical report, USDA Forest Service, Washington, DC.

K. N. Johnson & D. W. Rose, 1986. FORPLAN version 2: an overview. Technical report, USDA Forest Service Land Management Planning System Section, Washington, DC.

K. N. Johnson & H. L. Scheurman, 1977. Techniques for prescribing optimal timber harvest and investment under different objectives – Discussion and Synthesis. *Forest Science Monograph*, 18 (1977).

J. Jones, B. Meneghin, & M. Kirby, 1991. Formulating adjacency constraints in linear optimization models for scheduling projects in tactical planning. *Forest Science*, 37-5 (1991), 1283–1297.

M. Kirby, W. Hager, & P. Wong, 1986. Simulataneous planning of wildland management and transportation alternatives. *TIMS Stud. Management Science*, (1986).

S Kirkpatrick, 1983. Optimization by Simulated Annealing. *Science 220*, (1983), 671–680.

S. Kirkpatrick, 1984. Optimization by simulated annealing: Quantitative studies. *Journal of Stat. Physics*, 34 (1984), 875–986.

J. Lappi, 1992. A linear programming package for management planning. Research Papers 414, The Finnish Forest Research Institute, Suonenjoki.

P.E. Linehan & T.J. Corcoran, 1994. An expert system for timber harvesting decision making on industrial forest lands. *Forest products journal*, 44-6 (1994), 65–70.

C. Lockwood & T. Moore, 1993. Harvest scheduling with spatial constraints: a simulated annealing approach. *Canadian journal of forest research*, 23-3 (1993), 468–478.

lp_solve. Version 1.5, a LP and MIP solver written by Michel Berkelaar. Available for free from ftp://ftp.es.ele.tue.nl (131.155.20.126) at /pub/*lp_solve*.

A. K. Mackworth, 1977. Consistency in Networks of Relations. *Artificial Intelligence*, 8-1 (1977), 99–118.

B. Meneghin, M. Kirby, & J. Jones, 1988. An algorithm for writing adjacency constraints efficiently in linear programming models. In *The 1988 Symposium in systems analysis in forest resources*. USDA Forest Service General Tech. Report RM-161.

Steve Minton, Mark D. Johnston, Andrew B. Philips, & Philpip Laird, 1992. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58 (1992), 161–205.

G. Misund, B. S. Johansen, G. Hasle, & J. Haukland, May 1995. Integration of Geographical Information Technology and Constraint Reasoning - A Promising Approach to Forest Management. In *SCANGIS '95 - Proceedings of the 5th Research Conference on GIS, 12th-14th June 1995, Trondheim, Norway*, Jan Terje Bjø, rke, redactie.

U. Montanari, 1974. Networks of Constraints: Fundamental Properties of Applications to Picture Processing,. *Inform. Science*, 7 (1974), 95–132.

A.T. Murray & R.L. Church, 1995a. Measuring the efficacy of adjacency constraint structure in forest model. *Canadian Journal of Forest Research*, 25 (1995), 1416–1424.

A.T. Murray & R. L. Church, 1995b. Heuristic solution approaches to operational forest planning problems. *OR Spektrum*, 17 (1995), 193–203.

A. T. Murray & R. L. Church, 1994. Constructing and selecting adjacency constraints. *INFOR*, (1994). In Press.

D. I. Navon, 1971. Timber RAM– A long-range planning methods for commercial timber lands under multiple-use management. Technical report, USDA Forest Service, Research Paper PNW-70.

J. Nelson & J. D. Brodie, 1990. Comparison of random search algorithm and mixed integer programming for solving area-based forest plans. *Canadian Journal of Forest Research*, 20 (1990), 934–942.

J. Nelson, J. D. Brodie, & J. Sessions, 1991. Integrating short-term, area-based logging plans with long-term harvest schedules. *Forest Science*, 37-1 (1991), 101–122.

G. L. Nemhouser & L. A. Wosley, 1988. *Integer and Combinatorial Optimization.* New York, John Wiley and Sons.

C. H. Papadimitriou, A. A. Schaffer, & M. Yannakakis, 1990. On the complexity of local search. *Proc. 22nd Annual Symp. on Theory of Computing*, (1990), 438–445.

T. Pukkala & J. Kangas, 1993. A Heuristic Optimization Method for Forest Planning and Decision Making. *Scandanavian Journal of Forest Research*, 8 (1993), 560–570.

J. P. Roise, 1986. A nonlinear programming approach to stand optimization. *Forest Science*, 32 (1986), 735–748.

M. Siitonen, 1993. Experiences in the use of forest management planning models. *Silva Fennica*, 27-2 (1993), 167–178.

S. Snyder & C. ReVelle, 1996. The Grid Packing Problem: Selecting a Harvesting Pattern ina n Area with Forbidden Regions. *Forest Science*, 42-1 (1996), 27–34.

J.M. Torres-Rojo & J.D. Brodie, 1990. Adjacency constraints in harvest scheduling: an aggregation heuristic. *Canadian journal of forest research*, 20-7 (1990), 978–986.

E. Tsang, 1993. *Foundations of Constraint Satisfaction.* Harcourt Brace and Co.

P. C. Van Deusen, 1996. Habitat and harvest scheduling using Bayesian statistical concepts. *Canadian Journal of Forest Research*, 26 (1996), 1375–1383.

A. Wientraub, G. Jones, A. Magendzo, M. Meacham, & M. Kirby, 1994. A Heuristic System to Solve Mixed Integer Forest Planning Models. *Operations Research*, 42-6 (1994), 1010–1024.

A. Yoshimoto & J. D. Brodie, 1994. Comparative analysis of algorithms to generate adjacency constraints. *Canadian Journal of Forest Research*, 24-6 (1994), 1277–1288.

A. Yoshimoto, J. D. Brodie, & J. Sessions, 1994. A new heuristic to solve spatially constrained long-term harvest scheduling problems. *Forest Science*, 40-3 (1994), 365–396.

Appendix A

Data format for minconflicts

```
//first line
# of stands, number of periods, period length
//rest of the file
stand id, age, volume, trees#, tree height, area, neigh. -1, # of schedules(n)
// schedule 1
treatment type, first period, -1, age, volume, # trees, tree height, area
...
treatment type, last period, -1, age, volume, # trees, tree height, area
// schedule 2
...
// schedule n
...
stand id, age, volume, trees#, tree height, area, neigh. -1, # of schedules
...

Example:
The following is data for 1 stand, with 1 schedule with 10 periods
of length 5 years each.

1 10 5
2 25 15 0 4.54545 0 4 6 13 -1 1
0 1 -1 0 0 4.54545 0 0
0 2 -1 0 0 5.45455 0 0
```

```
1 3 -1 0 0 0 0 0
0 4 -1 0 0 0.909091 0 0
0 5 -1 0 0 1.81818 0 0
0 6 -1 0 0 2.72727 0 0
0 7 -1 0 0 3.63636 0 0
0 8 -1 0 0 4.54545 0 0
0 9 -1 0 0 5.45455 0 0
1 10 -1 0 0 0 0 0
```

Appendix B

Data Generator

```
float height (int yr, int a1, int a2, float h1, float h2);
int dbLoad (istream&, int&, const int, const int, const float, const float);

// the approximate function to calculate the height is as follows:
// case a: y = (h1/a1) * x
// case b: y = ((a2*h1 - a1*h2 + (h2-h1)) / (a2-a1)) * x
// case c: y = h2
// a1: age1 , a2: age2 , h1: height1 , h2: height2

//---------------------------------------------------------------------------
float height (int yr, int a1, int a2, float h1, float h2)
//---------------------------------------------------------------------------
{
    if (yr < a1) // case a
       return ((h1/a1) * yr);
    else if (yr <= a2) // case b
       return ( ((a2*h1 - a1*h2 + h2-h1) / (a2-a1)) * yr);
    else // case c
       return (h2);
}

//---------------------------------------------------------------------------
int dbLoad (istream& in, int& age, const int age_1, const int age_2,
```

```
            const float height_1, const float height_2)
//-------------------------------------------------------------------------
{
    int id, LastHarvesting, EarlyHarvesting,OptimalHarvesting,LateHarvesting;
    int maxVolYear;
    int rid;
    float TimeForGrowthTo2Meter, maxVol, Area;

    in >>  id;
    cout << id << " ";
    in >>  LastHarvesting;
    age = LastHarvesting * -1 ;
    cout << age << " ";
    in >>  TimeForGrowthTo2Meter;
    in >>  EarlyHarvesting;
    in >>  OptimalHarvesting;
    in >>  LateHarvesting;
    in >>  maxVolYear;
    in >>  maxVol;
    cout << 0 << " "; // # of trees
    cout << maxVol << " ";
    cout << height(age, age_1, age_2, height_1, height_2) << " ";
    in >>  Area;
    cout << Area << " ";
    in >> rid;
    while (rid != -1) {
        cout << rid << " ";
        in >> rid;
    }
    cout << "-1 ";
    return (1);

}

//-------------------------------------------------------------------------
int main(int argc, char* argv[])
//-------------------------------------------------------------------------
```

```
{
    int num, i, j, k, ok, age, origAge;
    char* file = argv[1];              // data file
    int MIN_AGE = atoi (argv[2]);      // for clearcut
    int MAX_AGE = atoi (argv[3]);      // for clearcut
    int age_1 = atoi (argv[4]);        // for height calc
    int age_2 = atoi (argv[5]);        // ...
    float height_1 = atof (argv[6]);   // ...
    float height_2 = atof (argv[7]);   // ...
    int nSch = atoi (argv[8]);         // number of schedules (domain size)
    int period = atoi (argv[9]);       // number of periods
    int pLength = atoi (argv[10]);     // period length

    if (argc < 10) {
        return (0);
    }
    ifstream in(file);
    in >> num;
    if ( !in.good() ) {
        return (-1);
    }
    cout << num << " " << period << " " << pLength << endl;
    for (i = 1; i <= num; i++) {
        if ( !in.good() ) {
            return (-1);
        }
        ok = dbLoad(in, age, age_1, age_2, height_1, height_2);
        if (ok != 1) {
            return (-1);
        }
        cout << nSch << endl;
        // now the "nSch" schedules
        origAge = age; int something = 0; int A[50];
        for (j = 1; j <= nSch; j++) {
            age = origAge;
            something = 0;
            for (k = 1; k <= period; k++) {
```

```
                if ( (age > (MIN_AGE + (j-1) * pLength)) &&
                   (age < (MAX_AGE + (j-1) * pLength)) ) {
                    A[k] = 1;
                    something = 1;
                    age = 0;
                }
                else {
                    A[k] = 0;
                }
                age = age + pLength;
            }

            if (something != 1) { // if no legal treatment found
                A[(random() % 9) + 1] = 1;
            }

            age = origAge;
            for (k = 1; k <= period; k++) {
                if ( A[k] == 1) {
                    cout << 1 << " " << k << " -1 ";
                    age = 0;
                }
                else {
                    cout << 0 << " " << k << " -1 ";
                }
                //cout << "0 0 0 0 0" << endl;
                cout << "0 0 ";
                cout << height(age, age_1, age_2, height_1, height_2) << " ";
                cout << "0 0" << endl;
                age = age + pLength;
            }
        }
    }
    return (1);
}
```

Appendix C

Test Application

```
//-------------------------------------------------------------------------
int main(int argc, char *argv[])
//-------------------------------------------------------------------------
{
    if (argc < 1) {
        return (0);
    }
    int maxStands = atoi(argv[1]);

    bool read = true;
    mcSchedule mySchedule;

    string dir = "~/development/src/app/minconf";

    read = mySchedule.dbLoad (dir, maxStands);
    mySchedule.solve();

    return(1);
}
```