Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 176 (2007) 1246–1264
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

O.R. Applications

Improving computational capabilities for addressing volume constraints in forest harvest scheduling problems

Juan Pablo Vielma [a,*], Alan T. Murray [b], David M. Ryan [c], Andres Weintraub [d]

[a] *Departamento de Ingeniería Matemática, Universidad de Chile, Blanco Encalada N° 2120, 5° Piso, Santiago, Chile*
[b] *Department of Geography, The Ohio State University, 1036 Derby Hall, 154 North Oval Mall, Columbus, OH 43210, USA*
[c] *Department of Engineering Science, University of Auckland, Private Bag 92019, Auckland, New Zealand*
[d] *Departamento de Ingeniería Industrial, Universidad de Chile, Casilla 2777, Santiago, Chile*

Received 17 August 2004; accepted 6 September 2005
Available online 18 November 2005

Abstract

Forest Harvest Scheduling problems incorporating area-based restrictions have been of great practical interest for several years, but only recently have advances been made that allow them to be efficiently solved. One significant development has made use of formulation strengthening using the Cluster Packing Problem. This improved formulation has allowed medium sized problems to be easily solved, but when restrictions on volume production over time are added, problem difficulty increases substantially. In this paper, we study the degrading effect of certain types of volume constraints and propose methods for reducing this effect. Developed methods include the use of constraint branching, the use of elastic constraints with dynamic penalty adjustment and a simple integer allocation heuristic. Application results are presented to illustrate the computational improvement afforded by the use of these methods.

Keywords: Combinatorial optimization; Branch and bound; Forest planning; Elastic constraints; Constraint branching

1. Introduction

The goal of the forest harvest scheduling problem is to select which areas of a forest will be harvested in different time periods. The forest is typically divided into small basic cells or management units, which

* Corresponding author. Present address: (PhD Student), School of Industrial and Systems Engineering, Georgia Institute of Technology, 765 Ferst Drive NW, Atlanta, GA 30332-0205, USA.
E-mail addresses: jvielma@isye.gatech.edu (J.P. Vielma), murray.308@osu.edu (A.T. Murray), d.ryan@auckland.ac.nz (D.M. Ryan), aweintra@dii.uchile.cl (A. Weintraub).

0377-2217/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2005.09.016

are then selected for harvesting. This problem has been frequently modeled as an integer programming problem whose objective is to maximize profits generated by the harvest schedule. With the intention of minimizing the environmental impact of these harvest schedules, regulations limiting the size of clear cut areas have been incorporated into the planning process (Thompson et al., 1973; Murray, 1999; Bettinger and Sessions, 2003). The restrictions imposing these conditions are known as maximum area restrictions and basically limit the contiguous area that can be harvested in a particular period or in a sequence of periods.

The basic cells delineating the forest are in general much smaller than the maximum area restriction. For this reason some groups of basic cells can be harvested together, creating many combinations of acceptable harvest schedules. One of the initial formulations for this problem, known as the Unit Restriction Model (URM) (Murray, 1999), simplifies the problem by aggregating basic cells into bigger cutting blocks. With the aid of geographic information systems (GIS), cutting blocks are constructed by adding adjacent basic cells until the combined area nears the maximum allowed. As a result of this a priori blocking, harvesting one block precludes neighboring blocks from being harvested in the same time period. Given this, a relatively straightforward adjacency or node packing problem may be structured.

It has been shown that if cutting block construction is incorporated into the decision process, then more profitable harvest schedules can be generated (Murray and Weintraub, 2002). One model that incorporates this block construction process into the problem is known as the Area Restriction Model (ARM) (Murray, 1999). Unfortunately, this model is very difficult to solve computationally. Recently, a strengthened formulation of the ARM, known as the Cluster Packing Problem, was developed in Goycoolea et al. (2005). This formulation incorporates the construction of cutting blocks by forming clusters of basic cells. Environmental area restrictions are addressed by limiting the area of each cluster and adding incompatibility restrictions. This approach leads to a set packing problem, and can be strengthened using adjacency relationships between basic cells. This formulation allows relatively large single period instances to be easily solved using commercial integer programming solvers. Many properties of this formulation are preserved for multi-period instances, but when volume production restrictions are added only instances with a small number of periods can be readily solved. This is likely caused by the fraction generating effect of typical volume constraints, which degrades the inherent integrality properties of the Cluster Packing Problem.

In this paper, we study several techniques to cope with this fraction generating effect. We first present constraint branching as a way of minimizing the damaging effects of strict volume constraints. We then discuss how strict volume constraints may not be the best way of modeling volume production requirements and propose elastic versions of these constraints as an alternative. This leads to the study of dynamic updating of penalties and an integer allocation heuristic as a way of assuring the generation of acceptable harvest schedules when elastic volume constraints are used. Finally, application results are provided to demonstrate the performance characteristics of the proposed approaches.

2. The forest harvest scheduling problem

As mentioned previously, harvest schedules generated by the area restricted model must comply with regulations that limit the contiguous area that can be harvested in a particular period or in a sequence of periods. As harvested sections of the forest are usually replanted immediately, the regeneration process dictates when subsequent additional harvesting can occur. This length of time is usually referred to as green-up. Although the Cluster Packing Problem can manage any length of green-up, one green-up period is used here for illustrative purposes. Consistent with the harvest scheduling literature, and forestry practice, when dealing with spatial harvesting decisions the planning horizon is such that any area of the forest can be harvested at most once.

The ARM can be structured using a graph representation of a forest region. In the following sections we describe this representation, show how to enforce maximum area limits and detail the Cluster Packing Problem approach for solving this problem.

2.1. Graph representation of a forest region

Data for the forest harvest scheduling problem is generally obtained from a GIS, where a forest is divided into *basic cells* as shown in Fig. 1.

For each cell we are given its area together with timber attributes, such as profit and volume, associated with harvesting in a particular time period.

Cells and adjacency relationships can be encoded in a graph $G(V, E)$ with node set V and arc set E. In this graph, each node is associated with a basic cell and an arc between two cells reflects adjacency. We consider two cells to be adjacent if they share a common border in the GIS. The graph associated with the forest shown in Fig. 1 is given in Fig. 2.

For each node u (or equivalently basic cell u), denote $\tilde{a}_u$ the area of cell u. Similarly, for each time period t denote $\tilde{c}_{u,t}$ and $\tilde{v}_{u,t}$ the profit and volume, respectively, of timber obtained if cell u is harvested in period t.

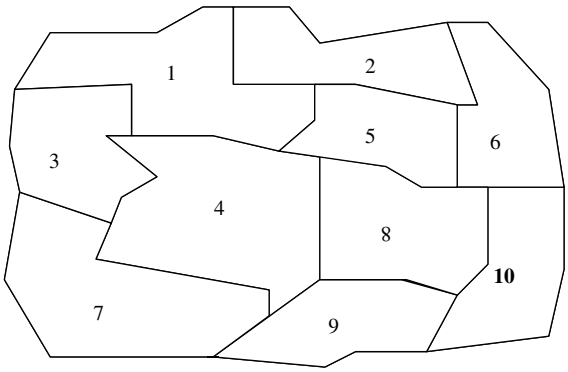

Fig. 1. Forest region partitioned into basic cells.

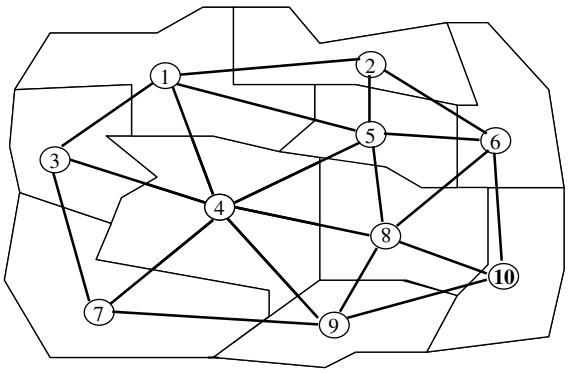

Fig. 2. Graph associated with forest region given in Fig. 1.

2.2. Feasible clusters and area restrictions

As mentioned earlier, a restriction that is typically incorporated in harvest scheduling is the maximum area restriction. This restriction limits the size of contiguous clear cut areas to a maximum of $A_{\max}$.

In most instances the area of a cell $u(\tilde{a}_u)$ is smaller than the maximum area ($A_{\max}$). Typical cell sizes range between 5 and 20 ha, with an associated maximum area of around 49 ha (Goycoolea et al., 2005). This means that some sets of contiguous cells may be harvested together, provided their combined area does not exceed $A_{\max}$. A *feasible cluster* is a set of contiguous cells that comply with the maximum area restriction. In other words, $S \subseteq V$ will be a feasible cluster if:

1. S induces a connected sub-graph in $G(V, E)$.
2. $\sum_{u \in S} \tilde{a}_u \leqslant A_{\max}$.

For example, if in Fig. 3 $\tilde{a}_5 + \tilde{a}_6 \leqslant A_{\max}$, then $\{5, 6\}$ is a feasible cluster.

Associated with each cluster there is information on area, profit and volume:

- $a_S = \sum_{u \in S} \tilde{a}_u$.
- $c_{S,t} = \sum_{u \in S} \tilde{c}_{u,t}$.
- $v_{S,t} = \sum_{u \in S} \tilde{v}_{u,t}$.

Two clusters are compatible if they are not adjacent and they do not share a common cell. For example, if given the feasible clusters in Fig. 3 of $\{2, 5\}$, $\{3, 7\}$ and $\{10\}$, then they are all compatible because no common cell or adjacency condition exists among these sets. However, given sets $\{1\}$ and $\{2, 5\}$, an adjacency condition would prohibit simultaneously harvesting these two sets. Thus, they are incompatible.

Because a green-up of one period is assumed, to comply with area restrictions we only need to forbid incompatible clusters from being harvested in the same period. The restrictions needed to enforce such conditions are denoted cluster incompatibility restrictions.

2.3. The cluster packing problem

The Cluster Packing Problem (CPP) is used here to maximize the net present value associated with selecting compatible clusters to be harvested. The CPP is modeled as a set packing formulation in which binary

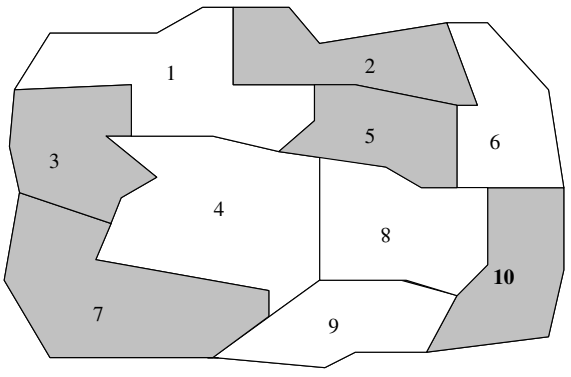

Fig. 3. Three compatible clusters.

variables indicate whether a cluster is harvested in a particular period. This formulation is given as follows using simple cluster incompatibility restrictions:

Cluster Packing Problem 1 (CPP1)

$$\text{Maximize} \sum_{S,t} c_{S,t} x_{S,t} \tag{1}$$

subject to

$$x_{S,t} + x_{S',t} \leqslant 1 \quad \text{for each pair of incompatible clusters } S, S' \text{ and for each period } t, \tag{2}$$

$$\sum_{S,t\ u \in S} x_{S,t} \leqslant 1 \quad \text{for each cell } u, \tag{3}$$

$$x_{S,t} \in \{0,1\} \quad \text{for each cluster } S \text{ and for each period } t, \tag{4}$$

where

- $x_{S,t} = 1$ if cluster S is harvested in period t.
- $c_{S,t}$ is Net Present Value of cluster S for period t.

The objective (1) of the CPP1 is to maximize the net present value of the selected forest harvest schedule. Constraints (2) are cluster incompatibility restrictions that make sure the maximum area limits are maintained. Constraints (3) enforce that each cell can only be harvested once in the planning horizon by allowing at most one cluster that contains a particular cell to be harvested in the planning horizon. Note that constraints (3) are the only part of the CPP1 model where basic cells are specifically considered as the model does not contain variables associated with basic cells, except for the particular case were a cluster consists of only one cell, i.e., when some cluster S is such that $S = \{u\}$ for some basic cell u. Finally, variables are required to be binary in constraints (4).

This formulation was proposed in Goycoolea et al. (2005) and the cluster incompatibility restrictions were strengthened using maximal cliques to give the problem better integrality properties. A clique is a complete sub-graph. For the forest graph a clique is a set of basic cells that are mutually adjacent to each other. A maximal clique is a clique that is not strictly contained in another clique. A strengthened version of CPP1 replaces Constraints (2) with the following:

$$\sum_{S \in \Lambda(K)} x_{S,t} \leqslant 1 \quad \text{for each maximal clique } K \text{ in graph } G(V,E) \text{ for each period } t, \tag{5}$$

where $\Lambda(K)$ is the set of all clusters that intersect maximal clique K. For more details regarding these constraints and the maximal cliques they are based on see Goycoolea et al. (2005).

Thus, CPP2 (Cluster Packing Problem 2) involves objective (1) subject to constraints (3)–(5).

3. Temporal volume constraints

Because of economic and legal reasons it is usually required that a "forest produces a non-declining even flow of timber" (Buongiorno and Gilless, 2003, p. 70) or a "reasonable yield pattern"(Ware and Clutter, 1971, page 436). Depending on the relationship between the discount rate applied and the growth rate of the forest, these requirements are difficult to address with certainty. For example, if the discount rate is much bigger than the growth rate of the forest and the profit per volume of timber before applying the discount rate is constant, the maximization of profits will generate a harvest schedule that will almost never give a non-declining flow of timber. The optimal schedule will harvest as much as allowed by the area restrictions in the first period and then harvest whatever is left as soon as allowed by the area restrictions in

successive period. This will usually generate a declining flow of timber, where no timber is available for harvest in later periods. Because of this behavior, non-declining flow requirements are typically modeled as linear constraints added to the harvest scheduling problem in order to ensure a consistent availability of timber. As the ARM is a harvest scheduling model, the previous discussion applies directly to the model we are studying, and hence these volume flow requirements must be included in some way. In this section, we will introduce a typical strict volume constraint, show some results for the Cluster Packing Problem incorporating these constraints and examine why these constraints make the problem difficult to solve.

3.1. Strict volume constraint

The non-declining requirement is added to ensure that timber flow is even or smooth. For this reason volume requirements are generally modeled as linear constraints that impose an overall non-declining pattern but do not enforce the non-declining requirement between periods. These volume constraints require that the volume of timber harvested in a period is within a Δ deviation of the volume harvested in the previous period:

$$(1-\Delta)\sum_{S} v_{S,t-1}x_{S,t-1} \leqslant \sum_{S} v_{S,t}x_{S,t} \leqslant (1+\Delta)\sum_{S} v_{S,t-1}x_{S,t-1}. \tag{6}$$

Note that these constraints are strict in the sense that the allowed Δ deviation is fixed, so any violation would be infeasible. If we add these strict volume constraints to the Cluster Packing Problem we get the following formulation:

CPP2-V (Cluster Packing Problem 2 – volume)

$$\text{Maximize} \sum_{S,t} c_{S,t}x_{S,t}, \tag{7}$$

subject to

$$\sum_{S\in\Lambda(K)} x_{S,t} \leqslant 1 \quad \text{for each maximal clique } K \text{ in graph } G(V,E) \text{ and for each period } t, \tag{8}$$

$$\sum_{S,t\ \ u\in S} x_{S,t} \leqslant 1 \quad \text{for each cell } u, \tag{9}$$

$$(1-\Delta)\sum_{S} v_{S,t-1}x_{S,t-1} - \sum_{S} v_{S,t}x_{S,t} \leqslant 0 \quad \text{for each period } t>1, \tag{10}$$

$$\sum_{S} v_{S,t}x_{S,t} - (1+\Delta)\sum_{S} v_{S,t-1}x_{S,t-1} \leqslant 0 \quad \text{for each period } t>1, \tag{11}$$

$$x_{S,t} \in \{0,1\} \quad \text{for each cluster } S \text{ and for each period } t. \tag{12}$$

This problem can be solved reasonably well for a very small number of periods (Goycoolea et al., 2005), but it is much harder to solve when many periods are considered. For example, Goycoolea et al. (2005) obtained an optimal solution for a single period problem with 1363 cells in 5 second. When the problem extended to 3 periods, 4 hour was needed to achieve a gap of 2.3%.

3.2. Computational results for model with strict volume constraints

Computational experiments were run using a forest in Northern California called El Dorado. Treatment periods were established at 10 years per period. Details for this region are given in Goycoolea et al. (2005), but the number of basic cells is 1363, the number of feasible clusters is 21,412, the number of maximal cliques is 2105, and the total number of constraints and variables for 15 periods (without volume constraints) are 32,938 and 321,180, respectively. Area and volume information are known and a fixed profit per volume

of timber was assumed. Multiple-period instances of 12 and 15 periods with annual discount rates of 3%, 6% and 8% were evaluated, assuming a green-up duration of one planning period. The runs were made on a Pentium 4 2 GHz PC with 2 Gb of RAM running Linux. Cplex 8.1 (Ilog, 2002) was used as the mixed integer programming (MIP) solver. Problem generation and additional programming was implemented using C++.

Our base test consisted of trying to solve the Cluster Packing Problem without volume constraints and with strict volume constraints using Cplex 8.1 directly. Table 1 gives computational results for the Cluster Packing Problem with strict volume constraints using $\Delta = 0.15$ and $\Delta = 0.10$. A time limit of 4 hour was imposed. The first two columns of Table 1 provide problem instance characteristics. The next two columns show allowable volume deviation and the number of branch-and-bound nodes processed. The next two columns present the time the best solution was found and the associated gap. The remaining columns provide solution attribute information.

Table 2 gives computational results for the Cluster Packing Problem without volume constraints. A time limit of 4 hour was imposed. The only difference between Tables 1 and 2 is the volume deviation in Table 1. In Table 2, the first two problem instances were able to be optimally solved using Cplex within the allotted 4 hour of processing. All other instances in Table 2 terminated with optimality gaps.

Table 1 shows that Cplex was not able to find integer feasible solutions for any problem instance when strict volume constraints are imposed. Alternatively, when the strict volume constraints are removed, all 6 instances could be solved, either optimally or close to optimal, within the 4 hour time limit. Furthermore, the first feasible solutions were found in minutes and solutions with an optimality gap (GAP) of less that 1% were found in less that 30 min for all instances without volume constraints. These results suggest that

Table 1
Results for the Cluster Packing Problem with strict volume constraints

Map/discount rate	Time periods	Δ	B&B nodes	Best solution time (second)	GAP (%)	1st sol under 1% GAP (second)	1st feasible time (second)	1st feasible GAP
El Dorado/3%	12	0.15	951	–	–	–	–	–
El Dorado/3%	15	0.15	696	–	–	–	–	–
El Dorado/6%	12	0.15	664	–	–	–	–	–
El Dorado/6%	15	0.15	754	–	–	–	–	–
El Dorado/8%	12	0.15	689	–	–	–	–	–
El Dorado/8%	15	0.15	1153	–	–	–	–	–
El Dorado/3%	12	0.10	764	–	–	–	–	–
El Dorado/3%	15	0.10	800	–	–	–	–	–
El Dorado/6%	12	0.10	699	–	–	–	–	–
El Dorado/6%	15	0.10	669	–	–	–	–	–
El Dorado/8%	12	0.10	0	–	–	–	–	–
El Dorado/8%	15	0.10	0	–	–	–	–	–

Table 2
Results for the Cluster Packing Problem without volume constraints

Map/discount rate	Time periods	B&B nodes	Best solution time (second)	GAP (%)	1st sol under 1% GAP (second)	1st feasible time (second)	1st feasible GAP
El Dorado/3%	12	3825	9983	Optimal	249	89	7.64
El Dorado/3%	15	5190	10,937	Optimal	308	136	7.66
El Dorado/6%	12	3235	12,841	0.28	1552	187	25.96
El Dorado/6%	15	3373	13,715	0.25	659	207	25.29
El Dorado/8%	12	2696	70,42	0.22	613	188	31.35
El Dorado/8%	15	2372	14,141	0.21	713	190	31.37

difficulty arises when strict volume constraints are imposed. This damaging effect is caused by the fractional generating influence the strict volume constraints have on the Cluster Packing Problem. As discussed previously, these constraints are added to ensure timber flow requirements. For this reason methods are needed for effectively dealing with these types of conditions.

4. Addressing volume constraints

The fractional generating effect of the strict volume constraints is explained by observing that the objective of maximizing net present value will generate optimal solutions to the linear relaxation at which about half of the volume constraints will be active. For example, if the discount rate applied to the objective function is bigger than the forest's growth rate, almost all of constraints (10) will be active to prevent the maximization of the objective function from harvesting as much as possible in the first period(s). As the coefficients of volume constraints contain many unrelated coefficients, it is very unlikely that volume constraints will be active for any integer solution. This behaviour causes the optimal solution to the relaxed model to usually have many fractions. In branch-and-bound, where fractions are being resolved, the optimization will continue to generate fractional solutions just to keep the volume constraints active in order to maximize the objective. This behavior will make it very difficult to find integer solutions with a linear relaxation based branch- and-bound procedure. Furthermore, the fact that there are many variables in a volume restriction (twice as many as there are feasible clusters) makes the generation of cover inequalities (Nemhauser and Wolsey, 1988), based on the knapsack interpretation of the volume restrictions, impractical. This fact also makes variable branching quite ineffective for finding integer solutions, as many *0-branches*(fixing a variable to 0) are needed to obtain integer solutions. We will elaborate on this later in the paper.

To eliminate some of the problems associated with strict volume constraints, we can either minimize their negative effect in branching or avoid compliance with them at equality. In this section, we study techniques that try to accomplish these objectives.

4.1. Constraint branching

The biggest problem with variable branching in the case of strict constraints is that many 0-branches are needed to get an integer solution. The fact that 0-branches will always be necessary to get an integer solution comes from the active volume constraints, as the LP relaxation will always comply with them exactly by fractionally harvesting some clusters. On the other hand, the fact that many 0-branches are needed comes from the weakness of a 0-branch, especially when many similar variables can take positive values. After most of the variables have been fixed by 1-branches, there will usually still be some volume of timber that can be harvested in a particular period while complying with the strict volume constraints. If this volume slack is smaller than the volume of any particular cluster, the LP relaxation will comply with the volume constraints exactly by fractionally harvesting some cluster. A 1-branch at this point in that particular cluster will make the problem infeasible so the only option for resolving that fraction is to follow the 0-branch. The problem is that there are many different clusters that can be harvested fractionally to use the volume slack, all of which will need to be fixed by 0-branches.

One of the main advantages of constraint branching is that it generates a much more balanced branch-and-bound tree, and hence does not generate weak branches (Ryan and Foster, 1981). Because of this behavior, constraint branching has frequently been used in cases where variable branching proved ineffective.

Constraint branching for the Cluster Packing Problem is based on the fact that any fractional solution must have at least one basic cell that is harvested fractionally in at least one of three ways: the cell is

harvested partially over several periods; the cell is harvested by multiple clusters in a particular period; or, the cell is harvested partially in one period and only by one cluster. By eliminating these three sources of fraction induction, integrality can be guaranteed.

In the following sub-sections, we describe three constraint branching approaches that eliminate each of the three possible fractions. They are referred to as *Cell/Time* branch, *Cell/Cell* branch and *Cell-Slack* branch. The latter can actually be viewed as a variable branching involving a slack variable.

4.1.1. Cell/time constraint branching

One way that a cell can be fractionally harvested is by being harvested partially over several periods. The cell/time constraint branch will eliminate this source of fractionality by forcing a particular cell to be harvested either before or after a given time period.

Let $\hat{x}_{S,t}$ be a solution to the linear relaxation of any of the CPP models at a given node in the branch-and-bound tree. If a cell u_0 is being harvested partially over several periods, then there must be a period t_0 such that:

$$0 < \sum_{t=0}^{t_0} \sum_{S \in \Lambda(u_0)} \hat{x}_{S,t} < 1, \tag{13}$$

where $\Lambda(u)$ is the set of all clusters that contain cell u.

For any t_0 that complies with (13), a constraint branch that will eliminate this fraction is to force cell u_0 to be harvested fully or not at all before period t_0. In other words, to force:

$$\sum_{t=0}^{t_0} \sum_{S \in \Lambda(u_0)} x_{S,t} = 1$$

in one branch and:

$$\sum_{t=0}^{t_0} \sum_{S \in \Lambda(u_0)} x_{S,t} = 0$$

in the other. Rather than viewing the implementation of each branch as the addition of the extra constraint, it is much better to implement the branch by removing the set of variables (i.e., assigning them a zero upper bound) which violate the constraint branch. That is, remove all variables harvesting cell u_0 after period t_0 in the first case and all those variables harvesting cell u_0 in periods up to and including t_0 in the second case.

Notice that if the branching is implemented by removing variables the possibility of cell u_0 not being harvested at all is included in both branches.

4.1.2. Cell/cell constraint branching

Another way in which a cell can be fractionally harvested is by being harvested by multiple clusters in a particular period. The cell/cell constraint branch will eliminate this source of fractionality by forcing two cells to be harvested either together or separately in a particular time period. Let $\hat{x}_{S,t}$ be a solution to the linear relaxation of any of the CPP models at a given node in the branch-and-bound tree. Suppose that cell u_0 is being harvested partially in period t_0 by two different clusters, S_1 and S_2. In other words $u_0 \in S_1 \cap S_2$ and $\hat{x}_{S_1,t_0} > 0 \wedge \hat{x}_{S_2,t_0} > 0$. Given that S_1 and S_2 are two different clusters and that cell u_0 belongs to both, there must be another cell $v_0 \neq u_0$ such that v_0 belongs only to one of the clusters. Without loss of generality, suppose that v_0 only belongs to S_1 (i.e. $v_0 \in S_1 \backslash S_2$). Then, a constraint branch that will eliminate this fraction is to forbid cells u_0 and v_0 from being harvested together in one branch and apart in the other. In other words, if $\Lambda(u)$ is the set of all clusters that contain cell u, then the first branch would be to force:

$$\sum_{S\in\Lambda(u_0)\cap\Lambda(v_0)} x_{S,t_0} = 0 \tag{14}$$

and the second would be to force:

$$\sum_{S\in(\Lambda(u_0)\cup\Lambda(v_0))\setminus(\Lambda(u_0)\cap\Lambda(v_0))} x_{S,t_0} = 0. \tag{15}$$

Again, it is better to implement these branches by fixing the corresponding variables to zero instead of explicitly adding the constraints.

4.1.3. Cell/slack constraint branching

In rare occasions, with strict or elastic volume constraints, it might happen that a cell will be harvested partially only in one period and by only one cluster. For this reason, we add an extra constraint branch to make sure that we can eliminate all possible fractions. This is called a cell/slack constraint branch.

Let $\hat{x}_{S,t}$ be a solution to the linear relaxation of any of the CPP models at a given node in the branch-and-bound tree. Suppose that cell u_0 is being harvested partially only in period t_0 and only by one cluster. In other words,

$$0 < \sum_t \sum_{S\in\Lambda(u_0)} \hat{x}_{S,t} < 1 \tag{16}$$

and no cell/time or cell/cell branch can be found.

In this case, the simple constraint branch forcing cell u_0 to be harvested fully or not at all would eliminate the fraction. More specifically the constraint branch would fix:

$$\sum_t \sum_{S\in\Lambda(u_0)} x_{S,t} = 1 \tag{17}$$

in one branch and:

$$\sum_t \sum_{S\in\Lambda(u_0)} x_{S,t} = 0 \tag{18}$$

in the other.

Notice that this constraint branch can be seen as a variable branch over the slack variable of the following restriction:

$$\sum_{S,t\ u_0\in S} x_{S,t} \leqslant 1. \tag{19}$$

Because of this fact, both branches can easily be implemented by fixing the corresponding slack to zero in the first branch and to one in the second.

4.2. Elastic volume constraints

As mentioned in Section 3.1, strict volume constraints (10) and (11) try to model the requirement that timber flows should be non-declining overall, but that this non-declining requirement does not need to be strictly enforced between all adjacent periods. Given this reasoning, if volume constraints are violated slightly they would still adequately model the smooth production requirement as long as these violations are not too big. Given that small violations are conceptually and theoretically valid, we could use elastic constraints to minimize the fraction generating effect of volume constraints. Elastic constraints have been used successfully in problems with similar fractional properties (Ehrgott and Ryan, 2003) and basically allow small violations to strict constraints while penalizing these violations in the objective function.

To implement elastic volume constraints in this harvesting problem we simply add continuous variables l_t, $u_t \geqslant 0$ that allow restrictions (10) and (11) to be violated:

$$(1-\Delta)\sum_S v_{S,t-1}x_{S,t-1} - \sum_S v_{S,t}x_{S,t} \leqslant l_t, \tag{20}$$

$$\sum_S v_{S,t}x_{S,t} - (1+\Delta)\sum_S v_{S,t-1}x_{S,t-1} \leqslant u_t. \tag{21}$$

These violations are then penalized in the objective function:

$$-\sum_{t>1}\underline{p}_t l_t - \sum_{t>1}\bar{p}_t u_t, \tag{22}$$

where penalties $\underline{p}_t, \bar{p}_t \geqslant 0$ are to be determined.

If we replace the strict volume constraints with these elastic versions, we get the following Cluster Packing Problem:

CPP2-EV (Cluster Packing Problem 2 – elastic volume)

$$\text{Maximize} \sum_{S,t} c_{S,t}x_{S,t} - \sum_{t>1}\underline{p}_t l_t - \sum_{t>1}\bar{p}_t u_t, \tag{23}$$

subject to

$$\sum_{S\in\Lambda(K)} x_{S,t} \leqslant 1 \quad \text{for each maximal clique } K \text{ in graph } G(V,E) \text{ and for each period } t, \tag{24}$$

$$\sum_{S,t\ u\in S} x_{S,t} \leqslant 1 \quad \text{for each cell } u, \tag{25}$$

$$(1-\Delta_E)\sum_S v_{S,t-1}x_{S,t-1} - \sum_S v_{S,t}x_{S,t} \leqslant l_t \quad \text{for each period } t>1, \tag{26}$$

$$\sum_S v_{S,t}x_{S,t} - (1+\Delta_E)\sum_S v_{S,t-1}x_{S,t-1} \leqslant u_t \quad \text{for each period } t>1, \tag{27}$$

$$x_{S,t} \in \{0,1\} \quad \text{for each cluster } S \text{ and for each period } t, \tag{28}$$

$$l_t, u_t \geqslant 0 \quad \text{for each period } t. \tag{29}$$

The objective (23) of the CPP2-EV can now be interpreted as maximizing the net present value of the selected forest harvest schedule while minimizing volume constraint violations. These violations, l_t and u_t, are defined by constraints (26) and (27). Notice that we have replaced Δ by Δ_E in the volume constraints to differentiate between the volume deviations used in strict and elastic versions of the constraints. The reason for this differentiation is that we might use different deviations for target strict volume constraints and elastic constraints used to enforce them. We will discuss this in detail in Section 4.2.1.

It should be noted that this technique can also be directly applied to other versions of volume constraints. For example, if instead of strict volume constraints (6) we used the following strict lower/upper bound volume constraints:

$$L \leqslant \sum_S v_{S,t}x_{S,t} \leqslant U \tag{30}$$

we could use the following elastic constraints:

$$L - \sum_S v_{S,t}x_{S,t} \leqslant l_t, \tag{31}$$

$$\sum_S v_{S,t}x_{S,t} - U \leqslant u_t. \tag{32}$$

Although reformulating the problem to include elastic versions of the volume constraints is straightforward, some technical details must be addressed for the procedure to be effective.

4.2.1. Using elastic constraints to comply with strict volume constraints

The main reason for using elastic constraints instead of strict volume constraints is that relative consistency in flow is really what is desired. The maximum acceptable violation will usually not be a fixed level and will probably need to be determined by practitioners that use the model. This argument implies that compliance with a particular strict volume constraint is not usually a main goal. Being able to fine tune parameters to keep violations controlled is an important goal and it is not immediately clear that this can be done while preserving the favorable properties of elastic constraints. A way to show that violations can be controlled is to demonstrate that compliance with a particular strict volume constraint can be achieved. We will do this in our computational results.

If we want to assure compliance with strict volume constraints (Eqs. (10) and (11)) with deviation Δ by using elastic constraints (Eqs. (26) and (27)) with deviation $\Delta_E = \Delta$, violations would need to be zero. This can be difficult to enforce and, as discussed in the Section 4.2.2, will destroy most of the favorable properties of elastic volume constraints. For this reason the best way to comply with a particular strict volume constraint with deviation Δ is to use elastic constraints with deviation $\Delta_E < \Delta$. In this way, complying with the strict volume constraints only requires that violations are kept small, but not necessarily equal to zero. This approach also has the advantage of penalizing violations before they are unacceptable, which further helps to control them.

4.2.2. Penalty value selection and control of violations

Selecting penalties that minimize the fractional generating effect of volume constraints while keeping the violations controlled may be very difficult. If big penalty values that force violations to be zero are selected, then solutions will be as fractional as they were using strict volume constraints. On the other hand, if small penalties are selected, violations may be unacceptably large. Ideally, we would like to start with very small penalties and slowly increase them to keep violations under control. To do this effectively the increment should be done during LP convergence to assure that optimal solutions to the LP relaxation will have controlled violations. Unfortunately, most commercial branch-and-bound solver do not allow this procedure. If we cannot effectively adjust the penalties we are forced to use bigger initial penalties to control violations. Fortunately, the fact that violations tend to increase as we descend in the branch-and-bound tree might help us avoid the problems associated with relatively big penalties. If we select the smallest penalties that will cause the root LP relaxation to have no violations, we can expect the violations to be positive deep in the branch-and-bound tree, where most of the integer solutions are found. Furthermore, if these violations do not grow too fast we can expect that these integer solutions will have acceptable violations. We shall see in the computational results that violations do usually remain controlled; however this is not always the case.

4.2.3. Dynamic adjustment of penalties using cuts

Most commercial branch-and-bound solvers do not allow the objective function to be modified during the branch-and-bound process. This prevents us from increasing the penalties after the LP relaxation has been solved.

Even though this is the case, we still have a way of dynamically updating penalties. It is possible to add a cut to the problem that will have the same effect as increasing a penalty. To do this penalty update through cuts, we just have to modify the formulation slightly.

Suppose that for variables $\mathbf{x},s \geqslant 0$ we have an elastic constraint of the form:

$$\mathbf{v}^T\mathbf{x} \geqslant s \tag{33}$$

and that the objective function coefficient associated with s is $-p_0$, where $p_0 > 0$. If we multiply inequality (33) by $p > 0$ we obtain the following equivalent inequality:

$$p \cdot (\mathbf{v}^{\mathrm{T}}\mathbf{x}) \leqslant p \cdot s \tag{34}$$

Replacing the right hand product $p \cdot s$ of (34) with a new variable $\lambda \geqslant 0$ we get the following inequality:

$$p \cdot (\mathbf{v}^{\mathrm{T}}\mathbf{x}) \leqslant \lambda. \tag{35}$$

If the objective coefficient associated with λ is equal to -1 then, in particular, inequality (35) with $p = p_0$ will have the same effect over variables $\mathbf{x}$ as the original inequality (33).

If we include variable λ with objective coefficient -1 in the original formulation, we can achieve the effect of an elastic constraint (33) with penalty level p by simply adding the cut (35). In particular, if we want to increase the penalty to a new level $q > p$ we only need to add the following cut:

$$q \cdot (\mathbf{v}^{\mathrm{T}}\mathbf{x}) \leqslant \lambda, \tag{36}$$

which will dominate the previous constraint when $\mathbf{v}^{\mathrm{T}}\mathbf{x} \geqslant 0$. This does not restrict generality, because when $\mathbf{v}^{\mathrm{T}}\mathbf{x} < 0$ any version of the elastic constraints will be inactive, i.e., the associated violation variable λ will be zero. Using this procedure we can modify the elastic model as follows:

CPP2-EVC (Cluster Packing Problem 2 – elastic volume with cut update)

$$\text{Maximize} \sum_{S,t} c_{S,t}x_{S,t} - \sum_{t>1} l_t - \sum_{t>1} u_t, \tag{37}$$

subject to

$$\sum_{S \in A(K)} x_{S,t} \leqslant 1 \quad \text{for each maximal clique } K \text{ in graph } G(V,E), \text{ for each period } t, \tag{38}$$

$$\sum_{S,t\ u \in S} x_{S,t} \leqslant 1 \quad \text{for each cell } u, \tag{39}$$

$$\underline{p}_t \left((1 - \Delta_E) \sum_S v_{S,t-1}x_{S,t-1} - \sum_S v_{S,t}x_{S,t} \right) \leqslant l_t \quad \text{for each period } t > 1, \tag{40}$$

$$\bar{p}_t \left(\sum_S v_{S,t}x_{S,t} - (1 + \Delta_E) \sum_S v_{S,t-1}x_{S,t-1} \right) \leqslant u_t \quad \text{for each period } t > 1, \tag{41}$$

$$x_{S,t} \in \{0,1\} \quad \text{for each cluster } S \text{ and for each period } t, \tag{42}$$

$$l_t, u_t \geqslant 0 \quad \text{for each period } t. \tag{43}$$

With this new formulation, increasing the penalties can be achieved by simply adding new restrictions. For example, if we want to increase the penalty associated with l_t from $\underline{p}_t$ to a new level $\underline{p}_t > \underline{p}_t$, the following restriction would be added as a cut:

$$\underline{q}_t \left((1 - \Delta_E) \sum_S v_{S,t-1}x_{S,t-1} - \sum_S v_{S,t}x_{S,t} \right) \leqslant l_t. \tag{44}$$

The procedure to increase penalties associated with violations u_t is analogous.

4.2.4. Fathoming invalid nodes

If we are using elastic constraints to solve a target strict volume constraint model, fathoming by infeasibility has to be treated with care. The problem is that a branch-and-bound node can be feasible for the elastic model and not for the target strict volume constraint model. These nodes can cause trouble with the penalty updating procedure.

It is possible that at some nodes in the branch-and-bound tree increasing the penalties will not be enough to control violations. This will happen when branching decisions that generate that node cause it to not contain any solution with controlled violations. In this case the node is actually infeasible for the target strict volume constraint, and hence can be fathomed.

To detect this case we can check the feasibility explicitly or try to detect when penalty increments do not affect the violations. A simpler way to fathom the nodes is to use the optimal objective value of the LP relaxation at that node. Given that the violations for this node will always be positive, increasing the penalties will eventually drive the objective function below the best incumbent solution, causing the branch-and-bound solver to automatically fathom the node. Of course, for this to work we have to assume that a feasible solution has already been found. In case it does not exist, we can simply postpone the fathoming until a feasible solution is found.

4.2.5. Integer allocation heuristic

Integer Allocation or Dive-and-Fix heuristics (Wolsey, 1998) are generally very effective when applied to set packing problems, like the Cluster Packing Problem without volume constraints. Unfortunately, when strict volume constraints are added, this kind of heuristic loses most of its effectiveness for the same reason that variable branching does.

The heuristic implemented is based on solutions to the LP relaxation of the elastic constraint model and simply fixes fractional variables to integer values. This process is repeated until all variables are integer. The heuristic also tries to keep the violations controlled, and corrects any significant violations.

As mentioned previously, depending on the relationship between the discount rate applied and the growth rate of the forest, either the first or last period will be the most profitable. The integer allocation heuristic starts fixing variables in this most profitable period first and then continues the variable fixing in adjacent periods. The heuristic only proceeds to the next periods if all fractions in the current period have been resolved. Further, before proceeding to the next period, the heuristic will conservatively try to correct most of the unacceptable volume constraint violations. Finally, when all variables have been fixed, the heuristic will aggressively correct unacceptable violations until there are no violations left. Fig. 4 contains the pseudo code describing the heuristic in detail. In Fig. 4 we assume that we are using elastic volume constraints model with Δ_E deviation to solve the strict volume constraint with Δ deviation and that $\Delta_E < \Delta$ as described in Section 4.2.1.

The performance of the heuristic depends on the elastic volume constraints. The fractional generating effect of the strict volume constraints will likely require many 0-branches in a branch-and-bound procedure to generate an integer solution. As integer allocation heuristics are essentially the same as depth first search in the branch-and bound tree, a heuristic of this kind will usually need to fix many small fractional variables to 0 when strict volume constraints are used. When elastic constraints are used instead, only a few variables will likely need to be fixed to 0.

5. Computational results

In this section, we present computational results to illustrate how elastic constraints can improve performance and how violations can be kept controlled. As discussed in Section 4.2.1, to show that violations can be kept controlled we will show that the stronger requirement of always complying with a particular strict volume constraint can be achieved. This approach will also allow us to compare solutions for models with strict volume constraints and models with elastic volume constraint as all solutions will be feasible for a target strict volume constraint model. This target model will be the strict volume constraint model CPP2-V with a deviations Δ equal to 0.10 and 0.15 for the volume constraints. When using the elastic constraint model CPP2-EV only solutions that comply with the corresponding strict volume constraint model

Variables:

$\hat{x}_{S,t}$: value of variable $x_{S,t}$ in the optimal solution to the last linear relaxation solve

$V_t := \sum_{S \in \Lambda} \hat{x}_{S,t}$: which is actualized every time $\hat{x}_{S,t}$ is actualized after an LP solve or by variable fixing

begin

Solve the linear relaxation of the elastic constraint model with Δ_E deviation and get $\hat{x}_{S,t}$

for $\hat{t} = 1$ **to** T **do**

Variable Fixing

while $\exists \ S \in \Lambda, \ \hat{x}_{S,\hat{t}} \in (0,1)$ **do**

M:= $\max_{S \in \Lambda} \hat{x}_{S,\hat{t}}$

if M>0.5 **then**

For all S such that $\hat{x}_{S,\hat{t}} \geq M - \varepsilon$ fix $x_{S,\hat{t}}$ to 1

else if M<0.5 **then**

Pick one S such that $\hat{x}_{S,\hat{t}} = M$ fix $x_{S,\hat{t}}$ to 1

else

Iteratively fix to 0 variables with $\hat{x}_{S,\hat{t}} < 0.5$ as long as $(1-\Delta)V_{\hat{t}-1} \leq V_{\hat{t}}$ and $(1+\Delta)V_{\hat{t}} \geq V_{\hat{t}+1}$

Resolve the linear relaxation and get $\hat{x}_{S,t}$

end while

Conservative Volume Corrections

if $V_{\hat{t}+1} < (1-\Delta)V_{\hat{t}}$ **or** $(1+\Delta)V_{\hat{t}-1} < V_{\hat{t}}$ **then**

Iteratively fix variables $\hat{x}_{S,\hat{t}}$ to 0 until $V_{\hat{t}+1} \geq (1-\Delta)V_{\hat{t}}$ and $(1+\Delta)V_{\hat{t}-1} \geq V_{\hat{t}}$ as long as the fixing does not make $(1-\Delta)V_{\hat{t}-1} > V_{\hat{t}}$ or $(1+\Delta)V_{\hat{t}} < V_{\hat{t}+1}$

end if

For all $S \in \Lambda$ fix $x_{S,\hat{t}}$ at its current value

end for

Final Aggressive Volume Corrections

while $\exists \ \hat{t} \ \ V_{\hat{t}+1} < (1-\Delta)V_{\hat{t}} \ \ or \ \ (1+\Delta)V_{\hat{t}-1} < V_{\hat{t}}$ **do**

Iteratively fix variables $\hat{x}_{S,\hat{t}}$ to 0 until $V_{\hat{t}+1} \geq (1-\Delta)V_{\hat{t}}$ and $(1+\Delta)V_{\hat{t}-1} \geq V_{\hat{t}}$

end while

Fig. 4. Integer allocation heuristic.

with Δ deviation will be accepted. Further, when calculating optimality gaps (GAP) the objective function of CPP2-V will be used, i.e., contributions by the penalties will not be considered, and the gaps will be cal-

culated with respect to the optimal value of the LP relaxation of CPP2-V with the corresponding Δ deviation. Elastic volume constraint deviations of $\Delta_E < \Delta$ will be used. More specifically, elastic volume constraint deviations will always be set to $\Delta_E = \Delta - 0.005$ where Δ is the deviation of the corresponding target strict volume constraint model.

Only results for the techniques and combinations of techniques which provided the best performance improvements for the Cluster Packing Problem are presented. In particular, it should be noted that although the dynamic adjustment of penalties is theoretically straight forward, penalty updating cuts produce numerical instabilities that caused them to perform poorly.

5.1. Strict model using constraint branching

Table 3 gives computational results for the Cluster Packing Problem with strict volume constraints using $\Delta = 0.15$ and $\Delta = 0.10$ using constraint branching. A time limit of 4 hour was imposed. The first two columns of Table 3 define the problem instance characteristics. Columns three and four show allowable volume deviation and the number of branch-and-bound nodes processed. The next two columns, present the time the best solution was found and the associated gap. Column seven shows the time it took to find a feasible solution under 1% gap and the last two columns show the time required for finding the first integer solution along with the optimality gap.

Constraint branching allows us to find either one or two very similar feasible solutions for each instance in the allotted time. The reason for this is that constraint branching can more effectively deal with the fractional properties of the strict volume constraints. Although constraint branching helps in finding feasible solutions for all instances, the quality of these solutions is not particularly good. This can probably be improved by a better selection between constraint branching alternatives or by a better selection of the next branch-and-bound node that is processed.

5.2. Elastic model with fixed penalties

Table 4 presents computational results for the elastic volume constraint model. Penalties for the elastic constraints were simply set to a fixed value giving no violations in the root LP relaxation. The only way in which we guaranteed compliance with the corresponding strict volume constraint was by only accepting valid solutions (solutions that are feasible for the corresponding target CPP2-V model). Constraint

Table 3
Results for the Cluster Packing Problem with strict volume constraints (constraint branching)

Map/discount rate	Time periods	Δ	B&B nodes	Best solution time (second)	GAP (%)	1st sol under 1% GAP (second)	1st feasible time (second)	1st feasible GAP
El Dorado/3%	12	0.15	3491	10,706	5.61	–	10,706	5.61
El Dorado/3%	15	0.15	2664	13,850	14.19	–	13,850	14.19
El Dorado/6%	12	0.15	3307	10,951	5.68	–	10,951	5.68
El Dorado/6%	15	0.15	2674	13,965	14.96	–	13,946	14.96
El Dorado/8%	12	0.15	3468	10,412	2.73	–	10,412	2.73
El Dorado/8%	15	0.15	2625	13,751	11.34	–	13,751	11.34
El Dorado/3%	12	0.10	3654	10,434	4.72	–	10,434	4.72
El Dorado/3%	15	0.10	3055	12,547	10.73	–	12,547	10.73
El Dorado/6%	12	0.10	3969	10,137	7.17	–	10,133	7.17
El Dorado/6%	15	0.10	2971	12,411	7.51	–	12,395	7.57
El Dorado/8%	12	0.10	3627	10,465	5.46	–	10,465	5.46
El Dorado/8%	15	0.10	2812	13,329	12.00	–	13,329	12.00

Table 4
Results for the Cluster Packing Problem with elastic volume constraints

Map/discount rate	Time periods	Δ	B&B nodes	Best solution time (second)	GAP (%)	1st sol under 1% GAP (second)	1st feasible time (second)	1st feasible GAP
El Dorado/3%	12	0.15	942	2424	0.56	2424	2424	0.56
El Dorado/3%	15	0.15	515	13,683	0.97	13,683	13,683	0.97
El Dorado/6%	12	0.15	604	6018	1.03	–	1075	32.09
El Dorado/6%	15	0.15	507	–	–	–	–	–
El Dorado/8%	12	0.15	634	2912	1.09	–	1128	1.14
El Dorado/8%	15	0.15	800	–	–	–	–	–
El Dorado/3%	12	0.10	687	3924	0.70	3924	3924	0.70
El Dorado/3%	15	0.10	687	13,621	1.13	–	13,621	1.13
El Dorado/6%	12	0.10	525	4683	1.03	–	4683	1.03
El Dorado/6%	15	0.10	346	9988	1.37	–	9988	1.37
El Dorado/8%	12	0.10	490	5699	1.32	–	5699	1.32
El Dorado/8%	15	0.10	920	6239	1.51	–	2532	1.70

branching was not used in this section. A time limit of 4 hour was imposed. The format of Table 4 is the same as Table 3.

Using elastic constraints directly, it is possible to find good valid feasible solutions for almost all instances. Still, in two cases it is not possible to find valid feasible solutions at all.

Solutions presented in Table 4 are for valid solutions that comply with the strict volume constraints, but many more feasible solutions were found. These were not considered as they were only feasible to the elastic volume constraint model and not for the target strict volume constraint model. In particular, for both instances in which no valid solutions were found, more than a hundred integer feasible solutions were found. Taking this into account, elastic volume constraints make it possible to find integer feasible solutions for all instances, but we cannot guarantee that all feasible solutions will be valid. By using elastic constraints we have traded the difficulty of finding integer feasible solutions for the difficulty of complying with the strict volume constraints. The fact that complying with the strict volume constraints was a problem only for two instances suggests this is a much smaller complication. Furthermore, the fact that for these two instances several invalid, but integer feasible, solutions were found suggests that we might be able to solve all instances if we can control the violations during the branch-and-bound process. On the other hand, out of these hundreds of integer feasible but invalid solutions, many almost complied with the target strict volume constraint model, so likely would be acceptable in planning situations.

5.3. Elastic model using the integer allocation heuristic and constraint branching

Table 5 presents computational results for the elastic constraint model using the integer allocation heuristic and constraint branching. Penalties for the elastic constraints were simply set to a fixed value that resulted in no violations in the root LP relaxation and the heuristic was executed at every node in the branch-and-bound tree.

Variable branching has the same effect as one step of the integer allocation heuristic, so using both of them together would be rather redundant. Further, constraint branching creates changes in the feasible space that are different to the variable fixings of the integer allocation heuristic, and hence diversify its greedy nature. For this reason we decided to use the constraint branching developed in Section 4.1 as the branching method in this section.

The heuristic always generates valid solutions (solutions that are feasible for the corresponding target CPP2-V model), so no extra validation was needed. A time limit of 4 hour was imposed.

Table 5
Results for the Cluster Packing Problem with elastic volume constraints (integer allocation heuristic)

Map/discount rate	Time periods	Δ	B&B nodes	Best solution time (second)	GAP (%)	1st sol under 1% GAP (second)	1st feasible time (second)	1st feasible GAP
El Dorado/3%	12	0.15	12	2095	0.72	2095	2095	0.72
El Dorado/3%	15	0.15	9	8998	0.82	7491	4443	1.25
El Dorado/6%	12	0.15	12	12,945	1.07	–	2454	1.34
El Dorado/6%	15	0.15	9	5808	1.22	–	4206	1.58
El Dorado/8%	12	0.15	17	4567	1.21	–	3255	1.27
El Dorado/8%	15	0.15	10	4003	1.64	–	4003	1.64
El Dorado/3%	12	0.10	11	2576	0.79	2576	2576	0.79
El Dorado/3%	15	0.10	11	9622	1.26	–	3938	2.13
El Dorado/6%	12	0.10	15	4252	1.26	–	3251	1.58
El Dorado/6%	15	0.10	11	4193	1.55	–	4193	1.55
El Dorado/8%	12	0.10	12	4494	1.43	–	3276	2.06
El Dorado/8%	15	0.10	9	8193	1.82	–	3488	1.93

It is clear in Table 5 that computational performance is better than using Cplex directly for the elastic constraint model. We can now find good valid solutions for all instances, even for the ones where Cplex applied directly to the elastic volume constraint model was not previously successful. These results are very encouraging when compared with our base case in which Cplex applied directly to the strict volume constraint model was not able to find any feasible solutions. Also note that, although the improvement is not large, the constraint branching does diversify the heuristic enough to get better solutions after the initial feasible solution.

6. Conclusions

In this paper, we have studied the effect of strict volume constraints in forest harvest scheduling using the Cluster Packing Problem. We have seen that the fractional generating effect of these strict volume constraints make the Cluster Packing Problem very difficult to solve. Several techniques have been introduced to minimize the negative effect of strict volume constraints: elastic versions of the volume constraints and constraint branching. By replacing the strict volume constraints with elastic versions, it is possible to find good feasible solutions for almost all problems, with some instances requiring comparatively less processing time. Although this is already an improvement compared to not finding any feasible solution in 4 hour, there still might be some problems associated with using elastic constraints directly. The main issue is how to make sure that strict constraint violations are controlled. One way of accomplishing this is by dynamically adjusting penalties associated with violations every time the violations grow too much. We have presented a way of implementing the dynamic adjustment of penalties that will work in most commercial branch-and-bound solvers. This implementation is theoretically straightforward, but some technical details still require further study. The main advantage of this approach is that implementation is not dependent on the specific model being solved. Hence, it can be used with any model that contains fractional generating constraints that could be slightly violated. Constraint branching allowed us to find feasible solutions for all test instances, but the solution quality was not very good. On the other hand, an interesting characteristic of constraint branching is that validity is maintained when volume constraints are modified, if new constraints are added or if the green-up is increased. Furthermore, constraint branching is able to generate feasible solutions even when strict volume constraints are used, so it may be useful if we do not want to worry about controlling violations. The strategy that proved most effective for finding integer feasible solutions with controlled violations was the addition of an integer allocation heuristic to the

branch-and-bound process. A simple variable fixing heuristic that managed violations found feasible solutions within 2% of optimality for all instances tested, with some instances within 1%. This comes a long way from not being able to find any feasible solutions when the strict model is solved directly and of course we can expect to get even better results if we are not forced do comply with a target strict volume constraint model.

Further research is needed for the dynamic adjustment of penalties to be computationally effective in all cases. In addition, it seems reasonable that the quality of solutions obtained by constraint branching could also be improved.

Acknowledgements

Funding for Murray was provided by the National Science Foundation (Geography and Regional Science Program and the Decision, Risk, and Management Science Program) under Grant BCS-0114362. Funding for Weintraub was provided by Fondecyt under Grants 1020317 and 1040520.

References

Bettinger, P., Sessions, J., 2003. Spatial forest planning: to adopt, or not to adopt? Journal of Forestry 101 (2), 24.

Buongiorno, J., Gilless, J.K., 2003. Decision Methods for Forest Resource Management. Academic Press, Elsevier Science, San Diego, CA.

Ehrgott, M., Ryan, D., 2003. The Method of Elastic Constraints for Multiobjective Combinatorial Optimization and its Application in Airline Crew Scheduling, Multi-Objective Programming and Goal Programming. Springer Verlag, Berlin, pp. 117–122.

Goycoolea, M., Murray, A., Barahona, F., Epstein, R., Weintraub, A., 2005. Harvest scheduling subject to maximum area restrictions: exploring exact approaches. Operations Research 53, 490–500.

Ilog, 2002. ILOG CPLEX 8.0 Users Manual.

Murray, A., 1999. Spatial Restrictions in Harvest Scheduling. Forest Science 45 (1), 1–8.

Murray, A., Weintraub, A., 2002. Scale and unit specification influences in harvest scheduling with maximum area restrictions. Forest Science 48, 779–789.

Nemhauser, G.L., Wolsey, L.A., 1988. Integer and Combinatorial Optimization. John Wiley and Sons, Inc., New York.

Ryan, D., Foster, B., 1981. An Integer Programming Approach to Scheduling. In: Wren, A. (Ed.), Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling. North Holland, Amsterdam, pp. 269–280.

Thompson, E.F., Halterman, B.G., Lyon, T.S., Miller, R.L., 1973. Integrating timber and wildlife management planning. Forestry Chronicle 47, 247–250.

Ware, G.O., Clutter, J.L., 1971. A mathematical programming system for the management of industrial forests. Forest Science 17, 428–445.

Wolsey, L.A., 1998. Integer Programming. John Wiley and Sons, Inc., New York.