Constraint Modeling for Forest Management

Eduardo Eloy, Vladimir Bushenkov, and Salvador Abreu

Abstract Forest management is an activity of prime economic and ecological importance. Managed forest areas can span very large regions and their proper management is paramount to an effective development, in terms both of economic and natural resources planning. A managed activity consists of individual and mutually independent policy choices which apply to distinct patches of land—named stands—which, as a whole, make up the forest area. A forest management plan typically spans a period of time on the order of a century and is normally geared towards the optimisation of economic or environmental metrics (e.g. total wood yield.) In this article we present a method which uses a declarative programming approach to formalise and solve a long-term forest management problem. We do so based on a freely available state-of-the-art constraint programming system, which we extend to naturally express concepts related to the core problem and efficiently compute solutions thereto.

Keywords Constraint programming · Forest management · Spatial restrictions · Adjacency constraints

1 Introduction

Forest management is a multi-faceted resource management and planning problem domain, one in which aspects such as the interests of multiple stakeholders, a diversity of underlying biological and physical models, economic performance and the

E. Eloy
University of Évora, Evora, Portugal
e-mail: m47215@alunos.uevora.pt

V. Bushenkov (✉)
CIMA, University of Évora, Evora, Portugal
e-mail: bushen@uevora.pt

S. Abreu
NOVA-LINCS, University of Évora, Evora, Portugal
e-mail: spa@uevora.pt

T. V. Tchemisova et al. (eds.), *Dynamic Control and Optimization*,
Springer Proceedings in Mathematics & Statistics 407,
https://doi.org/10.1007/978-3-031-17558-9_10

impact of climate change in more than one sense, all combine to create complex combinatorial optimisation situations.

Traditional Operations Research techniques, relying on Mixed Integer Linear Programming (MILP), while effective at solving a problem once it has been modeled, remain very difficult from a technical point of view. Constraint Programming [1, 2] provides a paradigm whereby one may directly model a problem in terms of the entities which are pertinent thereto, together with the relevant relations which they must observe, in a fairly abstract and generic fashion. This specification is understood to be *executable*, in that it is sufficient for a *constraint solver* to efficiently search for a solution. Moreover, because constraint programming relinquishes the notion of *control*, it stands as a natural candidate for the effective application to non-standard computing architectures, such as massively parallel systems, embodied in GPUs and hierarchical multiprocessors as found in HPC clusters with large-scale distributed memory [3]. This characteristic makes Constraint Programming potentially suitable for larger problem instances, as it not only facilitates the problem expression but also benefits directly from the increase in available computational resources, which is otherwise hard to exploit.

The remainder of this paper is structured as follows: the next section contains a brief revision of existing techniques used to solve similar problems. Section 3 recalls the basics of Constraint Programming. In Sect. 4 we provide a description of the model which was used to address the proposed problem. Section 5 contains a concise account of the prototype implementation, which is then experimentally evaluated in Sect. 6. Finally we conclude in Sect. 7 with a self-assessment and possible directions for further development.

2 Related Work

The use of forest resources is traditionally multifaceted. To ensure that these resources remain available for use by present and future generations, sustainable management practices are essential to balance the diverse and often competing demands of forest management. One way of planning management to minimise the impact of forestry activities is to include spatial constraints (or adjacency constraints) in the analysis of harvest schedules [4]. The objective function may be to maximise profit, net present value or other alternatives.

The Forest management authorities often place restrictions on the size of harvest openings. At present, the legal limits on clear-cut sizes in Portugal are set to 50 ha. Such restrictions can dramatically complicate the forest planning.

Many researchers used Integer Programming or Mixed-Integer Programming models to model spatial forest planning problems (see for example [5–8]). The decision variables usually correspond to harvesting blocks at a particular period of time. The Unit Restriction Model (URM) and Area Restriction Model (ARM) are the two main approaches to deal with adjacency in harvest scheduling models [9]. In the URM approach, the boundaries of each potential cutting block are predefined;

simultaneous harvesting is prohibited in two adjacent units. However, the ARM models allow simultaneous harvesting of adjacent units, provided their combined area does not exceed the maximum allowable cut size [9]. In the latter approach, the harvest block boundaries are not predefined; instead, they are defined through models that determine all the potential harvesting blocks that satisfy a maximum allowable cut [8, 10]. Formulating and solving ARM models is significantly more difficult than formulating and solving URM models [11]. Unlike URM models, ARM models are very flexible and generate more useful possibilities of better-performing harvesting plans. The first ARM formulations encompass an exponential number of variables or constraints. One integer programming ARM with a polynomial number of variables and constraints—called Area Restriction with Stand-Clear-Cut variables (ARMSC)—was proposed in [12].

These models are typically implemented as very large mixed integer linear programming (MILP) problems that are difficult to solve. The branch and bound algorithm (BBA) is a general method of obtaining exact solutions to MILP problems [5, 12, 13]. Until recently, however, only relatively small or medium problems could be solved with this algorithm. Considerable effort has been therefore put into developing alternative methods of solving harvest scheduling models with adjacency constraints, including heuristics, such as Monte Carlo integer programming [14], simulated annealing [15], genetic algorithm [16], and dynamic programming [17].

In this article we will apply a propagation-based Constraint Programming technique to solve one ARM problem for the Vale de Sousa region in Portugal [18].

3 Constraint Programming

Constraint modeling is a declarative paradigm which is also executable, hence the term *Constraint Programming*. This topic has been thoroughly covered in the literature over the last couple of decades and is the focus of a significant and active research community. There are several introductory texts which cover Constraint Programming, see for example [1].

An application may be formulated as a *Constraint Satisfaction Problem* (CSP) $\mathcal{P}$, which consists of a triple (V, D, C) where V is a set of *variables*, D is a set of *domains* for the elements of V and C is a set of *constraints*, i.e. relations over $\mathcal{P}(\mathcal{D})$ which must hold. The nature of the domains for the variables (*Finite Domains*, Booleans, Sets, Real numbers, Graphs, etc.), together with the specific relations (i.e. the *Constraints*) greatly influence the class of problems and application areas for which Constraint Programming form a good match. The vocabulary of *built-in constraints* combined with *composition operators* results in a very feature-rich and expressive formalism, which is arguably closer to most application domains than traditional O.R. formulations.

A *Constraint Optimisation Problem* (COP) is like a CSP but we are also interested in minimizing (or maximizing) an *objective function*. To achieve this, one may equate the objective function to the value of a particular variable. It is then possible to solve

a COP by iteratively solving interrelated CSPs, involving the addition of a constraint which establishes an inequation between the analytical definition of the objective function and the previously found value.

The model for an application problem may be declaratively formulated as a CSP, which will form the specification for a *constraint solver* to find a solution thereto. Many successful approaches have been followed to solve CSPs, namely systematic search, in which variables see their domain progressively restricted and each such step triggers the reduction of the domains of related variables, as dictated by the *consistency* policy—these are in general designated as propagation-based constraint solvers and there are several ones, some being presented as libraries for use within a general-purpose programming language, such as Gecode [19] or Choco [20]. Others offer a domain-specific language (DSL) which may be used to model a problem and provide it as input to different solvers; such is the case for instance for MiniZinc [21] or PyCSP3 [22].

Another approach entails selecting an initial solution candidate and working a path towards an actual solution by means of an *iterative repair* algorithm. The latter forms the basis for several *local search* techniques, which may be generalised to related methods called *metaheuristics*. Solvers which derive the strategy used to guide the search from the specification of a CSP are called *constraint-based local search* solvers [23] and combine the convenience of a declarative problem formulation with the (relative) efficiency of an *anytime algorithm*.

Solvers exist for both propagation-based search and metaheuristic search, which exhibit high performance and the capacity to make use of parallel hardware to attain yet better performance, e.g. as discussed in [24, 25].

Constraint modeling allows one to express a problem by means of both simple arithmetic and logic relations, but also resorting to *global constraints*. These are instance-independent yet problem-class-specific relations, for which particular dedicated algorithms can be devised and encapsulated in a reusable specification component. Intuitively, a global constraint expresses a useful and generic higher-level concept, for which there is an efficient (possibly black-box) implementation. For instance, the `AllDifferent` constraint applies to a set of variables and requires them to take pairwise distinct values. It may be internally implemented in a naïve way by saying that each distinct pair of variables in the list must be different, or it may resort to a more specialized algorithm to achieve the same result more efficiently. The application programmer will benefit from the performance gain with no additional effort.

Global constraints have proved to be a fertile ground for effective research, over the years. A limited common set of global constraints has been presented in the XCSP^3-core document [26], which lists 20 such frequently used and generally useful constraints. This forms the basic vocabulary of XCSP^3-core, an intermediate representation for CSPs designed with the purpose of interfacing different high-level modeling tools with distinct specific constraint solvers.

4 Modeling Forest Management

A Forest Management problem over a geographical map consists of:

- A set of *management units* (we may sometimes use the acronym MU), or *stands*, which represent contiguous pieces of land. A stand has fixed and time-dependent attributes. In the former case we include surface area and adjacency information (w.r.t. other stands.) The latter includes the species which is planted and the management action which is planned at a given time.
- A set of *prescriptions* which apply to the stands. These are actions to be carried out in the context of each stand, in a given point in time, and they include *harvesting* (removing all the trees), *thinning* (cutting off some wood but leaving the trees in place) or just doing nothing.

The problem we are aiming to solve entails selecting a prescription for each stand in the map, over the entire planning period. This goal is further qualified by a criterion which is to be optimised for, for instance maximum total wood output.

We are given as input a set P of *prescription definitions* which describe a sequence of time-indexed actions to be applied to each management unit (u_i.) Each prescription p is a tuple of the form: $(uid, id, wt, wh, c, y, s)$ where:

uid is a management unit identifier: an integer value over the finite set of unit identifiers $\mathcal{U} \subset \mathbb{N}$.[1]
id is an integer which uniquely identifies the prescription. It ranges over the finite set of prescription identifiers $\mathcal{P} \subset \mathbb{N}$.
wt is the wood reward for thinning.
wh is the wood reward for harvesting.
c is the cork reward.
y is the year this operation applies to (as an offset from the starting year for the simulation).
s is the *species* of tree which is planted on this management unit, and therefore available for the next period.

Prescriptions represent a strategy for each management unit, which will apply for the duration of the simulation. There may be more than one possible prescription for each management unit, with the understanding that they are mutually exclusive, i.e. one must choose which prescription to apply to each management unit. In the data we are working with, the simulation is carried out over a period of 90 years, so the domain of the y variables is the range $\{1 \ldots 90\}$.

Moreover, we have a set U of management units, each of which is a tuple of the form $u = (uid, s, a)$, where:

[1] Note that both the management unit identifiers and the prescription identifiers are remapped from the original external arbitrary string representation, which is more complicated than what we have here.

– $uid \in \mathcal{U}$ is the management unit identifier, as above.
– $s \in \mathbb{R}^+$ is a surface area (obtained from a geographical outline), expressed in hectares.
– a is a set of adjacent management units, qualified with the length of the common border, i.e. a set of pairs (uid_a, l_a) where uid_a is another management unit identifier and l_a is the length (in meters) of the shared border between uid and uid_a.

We define the function $adj : \mathcal{U} \to 2^{\mathcal{U}}$, which maps a management unit identifier to the set of its adjacent units. We also introduce an auxiliary collection of parameters, A_{ij} which are defined as $A_{ij} =$ **true** if $j \in adj(i)$ and **false** otherwise.

We also introduce the function $prescr : \mathcal{U} \to 2^{\mathcal{P}}$, which associates a unit identifier with the set of prescription identifiers which may apply to it. Conversely, we define the function $units : \mathcal{P} \to 2^{\mathcal{U}}$ which maps a prescription identifier to the set of unit identifiers it may apply to.

We model the problem as a Constraint Satisfaction Problem (CSP) over Finite Domains (FD).

Let P_i be the prescription assigned to management unit $i \in \mathcal{U}$, P_i will range over the set of identifiers for all prescriptions $\mathcal{P}$. We say S is a *well-formed solution* to the planning problem if it is a complete assignment to P_i, $\forall i \in \mathcal{U}$ which also satisfies the constraint:

$$\forall i \in \mathcal{U}, P_i \in prescr(i) \tag{1}$$

Besides stating what constitutes a well-formed solution, we want to express further restrictions on admissible solutions. A case in point is the total contiguous harvested area limit constraint, which may be formulated as:

> *For all management units, whenever there is a time in which, under the selected prescription, the management unit is to be totally harvested, then the sum of the areas of the adjacent management units (and closure thereof) which are also bring totally harvested, must be less than a preset limit ω.*

To help in meeting this goal, we introduce a function which captures the concept, which we call $glade : \mathcal{U} \times \mathbb{N} \to 2^{\mathcal{U}}$. This function is defined in terms of an auxiliary function g:

$$glade(i, t) = g(i, t, \varnothing)$$

The auxiliary function g is of type $g : \mathcal{U} \times \mathbb{N} \times 2^{\mathcal{U}} \to 2^{\mathcal{U}}$ and is recursively defined as follows:

$$g(i, t, I) = \begin{cases} I, & \text{if } i \in I \vee wh_{i,P_i,t} = 0 \\ \bigcup\limits_{j \in adj(i)} g(j, t, I \cup \{i\}), & \text{otherwise} \end{cases}$$

Intuitively, $glade(u, t)$ is the set of management units which are being harvested, contiguous with u.

With these definitions, we may specify the *harvested area limit* constraint:

$$\forall i \in \mathcal{U}, \forall t \in times(i), \left(\sum_{j \in glade(i,t)} area(j) \right) < \omega \quad (2)$$

With constraints (1) and (2) we are guaranteed to produce only solutions which respect the limit on contiguous area harvesting.

5 Implementation

Our initial implementation was carried out using the `Choco` Constraint Solver framework [20] and written in the Java programming language.

The process of implementing the problem entails firstly setting up 2 data structures, one is an array containing all the information of each MU obtained from the input called `Nodes`, and the other is an array of possible values to represent the possible prescriptions to apply to each MU called `MUS`. Note that `MUS` is a constraint variable array so the domain of possible values for each variable will be reduced until a solution is found. Another constraint variable array is called `WoodYields`, similar to `MUS` each index of the array represents something about the solution, in this case they represent the wood yielded by harvesting/thinning that MU when the corresponding prescription found in MUS is applied (this process involves using the "Element" constraint from the Choco-Solver framework), this is done so that at the end the contents of this array can be summed up to obtain the total amount of wood yielded by the solution. Once the setup is done we iterate through the main loop, as shown in Listing 1.1.

Listing 1.1 Main Loop

```
for (Var node in Nodes) {
    if (node.isValid()) {
        CreateConstraint (node, MUS).post();
        AddWoodToArray (node, MUS, WoodYields).post();
    }
}
```

This loop iterates through every `MU` in the input and imposes all valid constraints pertaining to it and its possible prescriptions. These constraints are implemented as a global constraint, via a custom propagator, as shown in Listing 1.2.

The propagator essentially iterates through the MU's possible prescriptions and checks if a prescription value can be applied by recursively checking its neighbouring MUs and their possible prescriptions. If at any one point the total sum of contiguous forest area cut down exceeds the given limit, the propagator fails and another value will be chosen. The propagator calls a recursive function which verifies that a given MU is valid, w.r.t. the maximum cut area requirement, as shown in Listing 1.3.

Listing 1.2 Custom Propagator

```
void propagate () {
    if (node.isValid() & node.hasCut()) {
        for (int year in node.yearsWithCuts) {
            try {
                gladePropagate(node, year, 0)
            } catch (Exception limitSurpassed) { fails() }
        }
    }
}
```

Listing 1.3 Propagator Helper

```
int gladePropagate (node, year, sum) {
  if (node.hasCut()) {
    sum += node.area;
    if (sum > MAXLIMIT) { throw Exception }
    for (neighbourNode in node.neighbours()) {
      if (neighbourNode.isValid()) {
        sum = gladePropagate (neighbourNode, year, sum);
      }
    }
  }
  return sum;
}
```

After leaving the main loop, the model has been fully setup. The Choco Solver framework is then told to set the objective of the solving to be one of *Maximisation*, and the target to be maximised is a constraint variable with the aforementioned sum of the contents in the `WoodYields` array.

Ultimately, the solver is activated and, if a solution is found then the resulting MU/prescription pairs are written to an output file.

6 Experimental Evaluation

The data for computational experiments is the same as in article [18] for the region of Vale de Sousa in the north of Portugal. The testing was done on a laptop running Ubuntu 20.04.3 LTS, with 4GB of available ram available and 4 cores. The code was compiled using java 8.

It should be noted that no solution for the full problem could be found in a reasonable time frame: the complete problem includes 1373 management units and the program ran for an entire day and a solution was not found. Consequently we opted for an approximation to the problem.

6.1 Limiting by Distance to Initial MU

By limiting the input and therefore the problem size it is possible to more closely observe the growth of the execution time.

To this end we decided to limit the number of MUs taken into account by initially choosing an "initial MU" and then only working with MUs within a maximum distance to that MU. So in the implementation the third argument is the Id of the "initial MU" and the fourth specifies a maximum distance, meaning only MUs within that distance to the "initial MU" are taken as valid input, the rest are ignored.

With this setup, while choosing the MU with Id `0` (which should be located near the Penafiel-PaivaNorte border) as the "initial MU" and gradually increasing the maximum distance by 1km at a time, solutions are found until the 5km distance mark, where the program finishes in around 4 min but does not find a solution (meaning that it is not possible to satisfy the constraints.) Increasing the distance to 6km results in the same outcome and by the 7km distance mark the program takes about 1.30–2h to finish execution and Fig. 1 shows the valid MUs in this context.

Once the 8km mark is reached we are dealing with 810 MUs, however this problem proves to be too complex and a solution is not found within a reasonable time frame. The median of 3 tests is shown in Table 1.

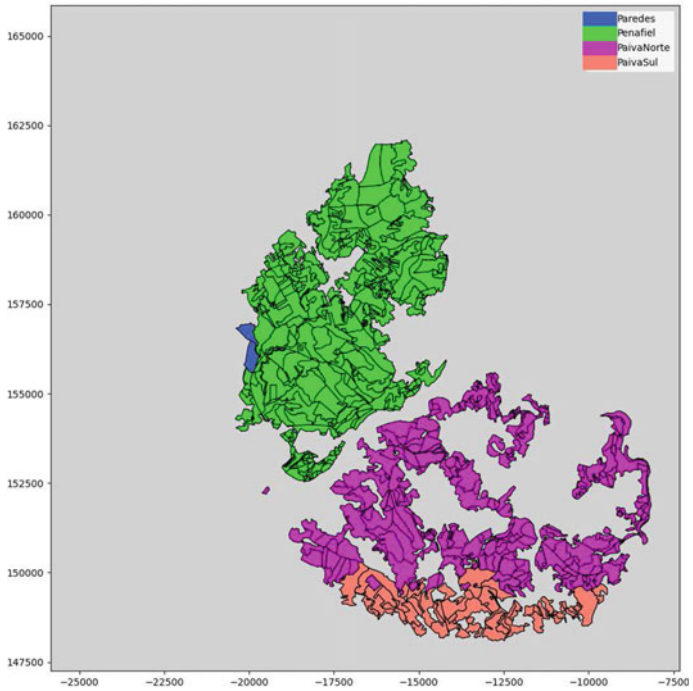

Fig. 1 MUs within a 7km distance from the "initial MU"

Table 1 Number of MUs, Runtime and Wood Yield for each step

Distance (km)	Number of MUs	Time (min)	Wood yield (kg)
1	21	0.162	58540
2	47	0.188	187724
3	103	0.253	557720
4	178	0.310	1042205
5	319	4.220	Solution not found
6	478	13.490	Solution not found
7	642	100.650	Solution not found
8	810	–	Solution not found

6.2 Restricting to a Sub-Region

Another way to limit the size of the problem is by selecting which of the 4 independent regions in Vale de Sousa should be managed, these are "Paredes", "Penafiel", "PaivaSul" and "PaivaNorte" and are shown in Fig. 2.

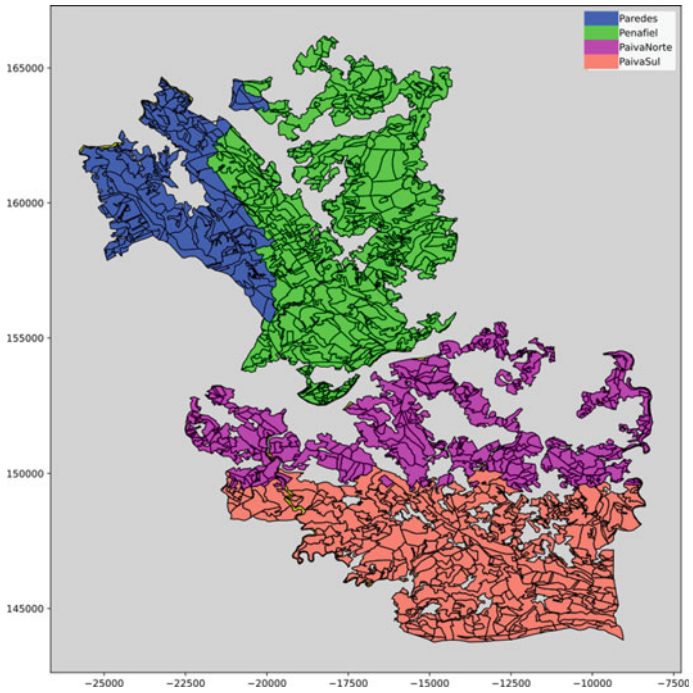

Fig. 2 Sub-Regions in Vale De Sousa

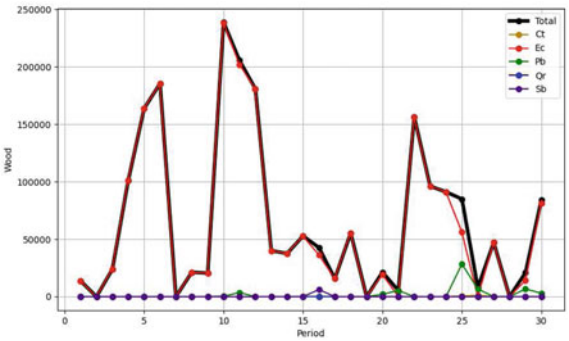

Fig. 3 Wood yield of a possible solution when applied to the Paredes region

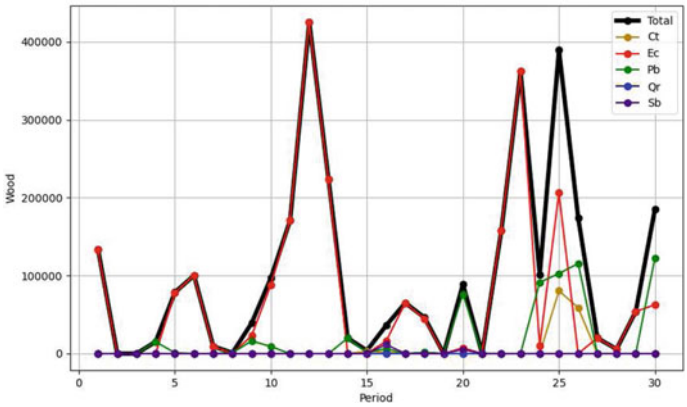

Fig. 4 Wood yield of a possible solution when applied to the PaivaNorte region

Paredes (Blue) The Paredes region of Vale De Sousa has 159 MUs, of which 18 have an area of over 40 ha, one of those being 49 ha. There is a total of 1998 possible prescriptions with 222 pertaining to the aforementioned 18 largest MUs. With the maximum limit set to 50 ha a solution is found in under a minute, this solution yield 1356 tonnes of wood (Figs. 3 and 4).

In the previous tests where the limit is based on maximum distance to the starting MU with internal Id 0, it's observable that the Paredes region doesn't factor in before the problem becomes unsolvable so it's possible to conclude that this region by itself is not problematic nor too complex for the implementation to handle.

PaivaNorte (Pink) This region has 351 MUs with only 8 of them being over 40 ha and a total of 7362 possible prescriptions. Predictably, if the assumption that the number of large MUs is a meaningful factor, a solution within the 50 ha limit is found in a reasonable time frame, yielding 2021 tonnes of wood. A large section of this region is always within the distance limits to the MU with Id 0 since that

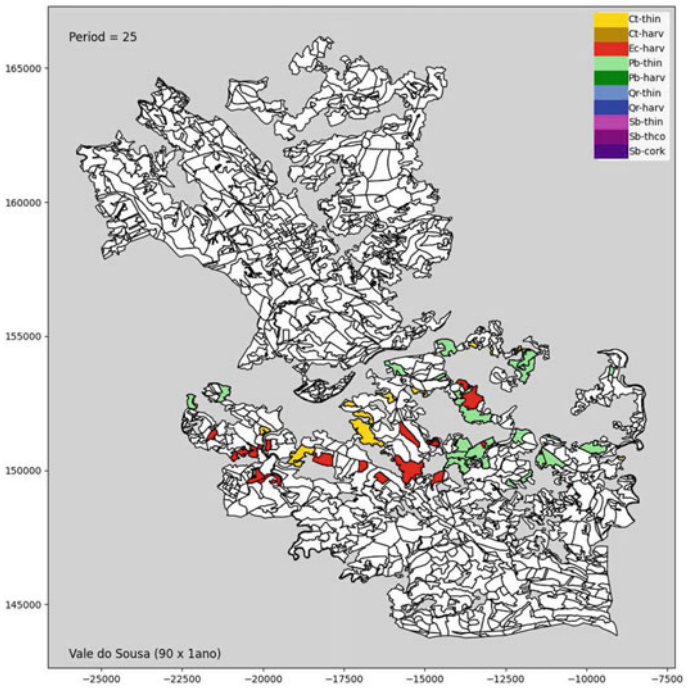

Fig. 5 Example of a solution provided by the implementation

MU is in PaivaNorte. Figure 5 provides an example of a solution outputted by the implementation, in this case the actions to take in PaivaNorte during period 25.

PaivaSul (Light Red) This region has 350 MUs, of which 30 have an area of over 40 ha and there are 17 instances where these large MUs are adjacent to each other, there is a total of 6998 possible prescriptions. The MU with internal Id 1133 belonging to this region is adjacent to 3 other large MUs, which would force the solver to find solutions where these MUs are never harvested in the same year.

If the limit is set to 50 ha a solution is not found in a reasonable time frame, by increasing this limit gradually the value with which the solver finds a solution in a similar time frame as the previous tests is 81 ha, which yields 3811 tonnes of wood.

The upper section of the PaivaSul region is within the 7km distance limit to the MU with Id 0, however when that distance limit is increased, some of the lower section of the region where more large MUs are located becomes valid input which adds to the complexity of the problem (Fig. 6).

Testing Penafiel and Paredes+PaivaNorte+PaivaSul separately By searching for solutions to the problem for the Penafiel region and the rest of Vale de Sousa separately some interesting results are obtained. The prototype was first run without the 455 MUs located in Penafiel, so only searching the 860 remaining MUs, and with the area limit set 50 ha. An optimal solution was found in 789 min (around 13 h),

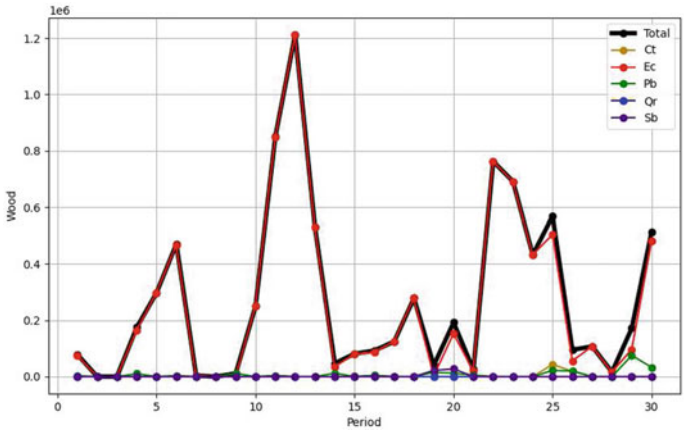

Fig. 6 Wood yield of a possible solution when applied to the PaivaSul region

obviously a large amount of time but when running the implementation with all the regions (including Penafiel) and the limit set to 50 ha a solution is not found within this time frame. This gives us a solution that maximises wood yield to the original problem but only for the Paredes, PaivaNorte and PaivaSul regions. Of note is that, since the Paredes region is separated from both Paivas, the optimal solution found is the same as the one obtained when searching only in the Paredes region, however the solution for the Paiva regions differs from the one found when searching them separately.

The next step was finding a solution only for the Penafiel region.

Penafiel(Green) This region has 455 MUs, of which 26 have an area of over 40 ha, 4 of which have 49 ha, and a total of 8593 possible prescriptions.

There are 13 instances of these large MUs being adjacent to each other, with adjacencies that seem to imply a clustering of around 9 of these large MUs. These factors add up to this region being the most problematic and possibly responsible for the dramatic increase in complexity when attempting to find a solution for the whole forest.

The limit had to be raised to 96 ha in order to find a solution in a reasonable time frame, which yields 3626 tonnes of wood (FIg. 7).

Increasing the limit of the area constraint Another useful test to analyze this implementation is to check for variations in total wood yield in the solutions found, when the maximum limit for contiguous forest area harvested in a year is increased. The regions picked for this test were the Paredes and PaivaNorte regions because they both have solutions when the area limit constraint is set to 50 ha.

Predictably, as shown in Tables 2 and 3, the wood yield tends to increase when the area limit constraint is increased because the solver attempts to find solutions that maximize that output but at some point the amount of wood yielded tends to plateau.

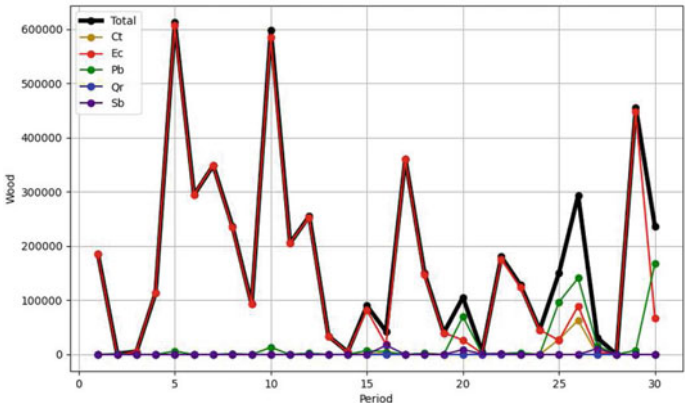

Fig. 7 Wood yield of a possible solution (not respecting the 50 ha limit) when applied to the Penafiel region

Table 2 Area limit and Wood Yield for each step, Paredes region

Limit (ha)	Wood yield (kg)
50	1356534
55	1363613
60	1366454
65	1377433
70	1393014
75	1412062
80	1411094
85	1415788
90	1418554
95	1417703

Table 3 Area limit and Wood Yield for each step, PaivaNorte region

Limitb (ha)	Wood yield (kg)
50	2021722
55	2035704
60	2042593
65	2044444
70	2049264
75	2049392
80	2073522
85	2083581
90	2082840
95	2086786

7 Conclusions and Future Work

In this project we proved that the Forest Management problem can be conveniently described as a Constraint Optimisation Problem, where all the selection criteria and optimisation objectives may be integrated without need for further changes or additions to the base model.

We contributed a new global constraint, which ensures a "flood filled" area surrounding a given point observes specific conditions, e.g. being under a general limit.

The implementation, although functional, is currently limited in its reach when applied to the full-scale problem, but it compares well to our previous design of a MILP solver. This is particularly striking because the problem we are solving is more general than that previously tackled in the literature: we allow for the cuts to happen independently and possibly more than once over the entire simulation period.

We are currently working on metaheuristic solvers, including local search, aiming to tap the inherent performance benefits of these methods when doing combinatorial optimisation. The solvers we are planning to use are designed to exploit parallel and distributed computing resources, which we expect to be a key factor in attaining better performance, with the potential to scale into the thousands of cores as witness [27]. An interesting aspect of these solvers is that they include *constraint-based local search* solvers, for which the problem specification is just a Constraint Optimisation Problem, as we currently have.

Acknowledgements This work was partly funded by Fundação para a Ciência e Tecnologia (FCT) under grants LISBOA-01-0145-FEDER-030391, PTDC/ASP-SIL/30391/2017 (BIOECOSYS), PCIF/MOS/0217/2017 (MODFIRE), strategic projects UIDB/04674/ 2020 (CIMA) and UIDB/04516/2020 (NOVA LINCS). Some of the experimental work was carried out on the `khromeleque` cluster of the University of Évora, which was partly funded by grants ALENT-07-0262-FEDER-001872 and ALENT-07-0262-FEDER-001876.

References

1. Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, volume 2 of Foundations of Artificial Intelligence. Elsevier (2006)
2. Krzysztof, R.: Apt. Principles of Constraint Programming. Cambridge University Press (2003)
3. Machado, R., Abreu, S., Diaz, D.: Parallel performance of declarative programming using a PGAS model. In: Sagonas, K (ed.) Practical Aspects of Declarative Languages - 15th International Symposium, PADL 2013. Rome, Italy. Proceedings, volume 7752 of Lecture Notes in Computer Science, pp. 244–260. Springer, Berlin (2013)
4. Bettinger, P., Boston, K., Siry, J., Grebner, D.; Spatial restrictions and considerations in forest planning. In: Forest Management and Planning, pp. 249–267. Academic (2017)
5. Gharbi, C., Ronnqvist, M., Beaudoin, D., Carle, M.-A.: A new mixed-integer programming model for spatial forest planning. Can. J. For. Res. **49**, 1493–1503 (2019)
6. Gunn, E., Richards, E.: Solving the adjacency problem with stand-centered constraints. Can. J. For. Res. **35**, 832–842 (2005)
7. Hof, J., Joyce, L.: a mixed integer linear programming approach for spatially optimizing wildlife and timber in managed forest ecosystems. Forest Sci. **39**, 816–834 (1993)

8. McDill, M.E., Rebain, S., Braze, M.E., McDill, J., Braze, J.: Harvest scheduling with area-based adjacency constraints. Forest Sci. **48**(4), 631–642 (2002)
9. Murray, A.: Spatial restrictions in harvest scheduling. Forest Sci. **45**(1), 45–52 (1999)
10. Goycoolea, M., Murray, A., Vielma, J.P., Weintraub, A.: Evaluating approaches for solving the area restriction model in harvest scheduling. Forest Sci. **55**(2), 149–165 (2009)
11. Baskent, E.Z., Keles, S.: Spatial forest planning: a review. Ecol. Model. **188**, 145–173 (2005)
12. Constantino, M., Martins, I., Borges, J.G.: A new mixed-integer programming model for harvest scheduling subject to maximum area restrictions. Oper. Res. **56**(3), 542–551 (2008)
13. McDill, M.E., Braze, J.: Using the branch and bound algorithm to solve forest planning problems with adjacency constraints. Forest Sci. **47**(3), 403–418 (2001)
14. Boston, K., Bettinger, P.: An analysis of monte carlo integer programming, simulated annealing, and tabu search heuristics for solving spatial harvest scheduling problems. Forest Sci. **45**(2), 292–301 (1999)
15. Borges, P., Eid, T., Bergseng, E.: Applying simulated annealing using different methods for the neighborhood search in forest planning problems. Eur. J. Oper. Res. **233**(3), 700–710 (2014)
16. Boston, K., Bettinger, P.: Combining tabu search and genetic algorithm heuristic techniques to solve spatial harvest scheduling problems. Forest Sci. **48**(1), 35–46 (2002)
17. Borges, J.G., Hoganson, H.M., Rose, D.W.: Combining a decomposition strategy with dynamic programming to solve the spatially constrained forest management scheduling problem. Forest Sci. **45**(1), 201–212 (1999)
18. Marques, S., Bushenkov, V., Lotov, A., Borges, J.G.: Building pareto frontiers for ecosystem services tradeoff analysis in forest management planning integer programs. Forests **12**, 1244 (2021)
19. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling. In: Schulte, C., Tack, G., Lagerkvist, M.Z. (eds.) Modeling and Programming with Gecode (2009). Corresponds to Gecode 6.2.0
20. Prud'homme, C., Fages, J.-G., Lorca, X.: Choco Solver Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2016)
21. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Bessiere, C. (ed.) 13th International Conference on Principles and Practice of Constraint Programming—CP 2007, Providence, RI, USA, Proceedings, volume 4741 of Lecture Notes in Computer Science, pp. 529–543. Springer, Berlin (2007)
22. Lecoutre, C., Szczepanski, N.: Pycsp3: Modeling combinatorial constrained problems in python (2020). arXiv:2009.00326
23. Michel, L., Van Hentenryck, P.: Constraint-based local search. In: Martí, R., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Heuristics, pp. 223–260. Springer, Berlin (2018)
24. Codognet, P., Munera, D., Diaz, D., Abreu, S.: Parallel local search. In: Hamadi, Y., Sais, L. (eds.) Handbook of Parallel Constraint Reasoning, pp. 381–417. Springer, Berlin (2018)
25. Régin, J.-C., Malapert, A.: Parallel constraint programming. In: Hamadi, Y., Sais, L. (eds.) Handbook of Parallel Constraint Reasoning, pp. 337–379. Springer, Berlin (2018)
26. Boussemart, F., Lecoutre, C., Audemard, G., Piette, C.: Xcsp3-core: a format for representing constraint satisfaction/optimization problems (2020). arXiv:2009.00514
27. Machado, R., Pedro, V., Abreu, S.: On the scalability of constraint programming on hierarchical multiprocessor systems. In: 42nd International Conference on Parallel Processing, ICPP 2013, pp. 530–535. IEEE Computer Society,Lyon, France (2013)