Multi-objective Finite-Domain Constraint-Based Forest Management

Eduardo Eloy, Vladimir Bushenkov, and Salvador Abreu

Abstract This paper describes an implementation of a Constraint Programming approach to the problem of multi-criteria forest management optimization. The goal is to decide when to harvest each forest unit while striving to optimize several criteria under spatial restrictions. With a large number of management units, the optimization problem becomes computationally intractable. We propose an approach for deriving a set of efficient solutions for the entire region. The proposed methodology was tested for Vale do Sousa region in the North of Portugal.

Keywords Forest management · Constraint programming · Constraint modeling · Multi-criteria optimization · Pareto frontier

1 Introduction

This paper presents an extension to previous work [1] of the authors on single criteria Forest Management optimization by considering multiple-criteria. Forest Management remains an activity of prime ecological importance where the interests of multiple stakeholders can lead to complex combinatorial and optimization problems, more so when different measures of economic performance such as wood yield and cash flow have to be balanced with environmental impact measures such as soil loss and fire resistance.

E. Eloy (✉)
University of Évora, Evora, Portugal
e-mail: eeloy@di.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

J. P. Almeida et al. (eds.), *Operational Research*, Springer Proceedings in Mathematics & Statistics 437, https://doi.org/10.1007/978-3-031-46439-3_6

Fig. 1 Vale De Sousa forest and its regions

Traditionally these problems are modelled and implemented with Integer Programming approaches and adjacency constraints [2–5], which work by applying conditions to when activities like harvesting can be applied to adjacent units of land. These approaches offer varying results depending on the model and the specific forest case. This paper describes an approach utilizing Constraint Programming (CP) [6] to solve a concrete problem of multiple-objective optimization, those objectives being wood yield, soil loss and fire resistance.

The effort to develop this approach was made possible by the MODFIRE project, which provided us with funding as well as all the relevant data regarding the forest which we based this implementation on, the Vale de Sousa forest, with the goal of optimizing multiple objectives when managing the forest throughout the 50 year planning horizon (2020-2070).

The forest is divided into 1406 Management Units (MUs), as shown in Fig. 1.

For each MU several different *prescriptions* were defined. Each prescription provides an option as to when each MU should be harvested (cut down every tree and making a clear-cut) or its branches thinned, as well as what "reward" is obtained by applying either of those actions in terms of the different optimizable criteria. It is assumed that in each MU there is only 1 species of tree which is specified by the prescriptions, with the possible tree species being eucalyptus (Ec), cork oak (Sb), pine tree (Pb), chestnut tree (Ct), pedunculate oak (Qr) and different riparian species of trees (Rp).

The main restriction regarding the harvest action is that to prevent issues of soil erosion the environmental authority imposes a limit to the *continuous amount of forest area that can be harvested (cut down)* and this limit is generally set at 50ha.

The combinatorial problem is thus as follows: to attribute a prescription to each MU so that in any given year, the continuous adjacent area of MUs where the action to be taken that year is a harvest must be below 50ha, meaning there cannot be very large sections of continuous forest area being deforested in any single year. To simplify the model, restrictions on wood flow volume were not considered.

The paper is structured as follows: after introducing the previous problem statement, we briefly review the state of the art in sect. 2 and then succinctly recap Constraint Programming (CP) in Sect. 3. In Sect. 4 we discuss the implementation and computational environment and evaluate the performance of the system in Sect. 5. Section 6 concludes this paper with a brief analysis and possible directions for further development.

2 Related Work

As previously stated the present work comes as a continuation of our participation in the BIOECOSYS project, as documented in [1], and of our current participation in the MODFIRE project. In that project we worked with a different structure for the Vale de Sousa Forest: the Paiva sub-region was divided into north and south and Penafiel included a cluster of several large MUs which made it a hard sub-region to solve. This was because a valid combination of prescriptions where none of the MUs composing these clusters were harvested at the same time was never found by the solver. Moreover, the planning horizon was 90 years instead of 50 years and the prescriptions only came associated with one optimizable criterion which was the wood yield.

The implementation we have now is more complete and able to also model and deal with multi-criteria optimization.

In other methods and models for forest planning problems, such as the ones cited in this paper's introduction, the authors usually represent the forests as adjacency graphs, so we did as well.

In a 1999 paper Alan T. Murray [7] proposed modeling constraints on the maximum harvest area by allowing multiple adjacent units to be harvested as long as their combined area does not exceed a pre-defined limit. We adopt this approach, termed Area Restriction Model (ARM), with a parametric area limit.

Latter Constantino et al. [8] proposed for this problem a compact mixed integer linear programming model, with polynomial number of variables and constraints. Multi-criteria and bi-level approaches have also been applied in forest management [9–11].

3 Constraint Programming

A good way to understand CP is that it stands as a synthesis of Mathematical Programming, where the programmer specifies the problem and the system attempts to find a solution on its own, and Computer Programming, where the programmer has to instruct the system on how to go about finding a solution. In CP the programmer specifies the problem as a set of relations that must hold (not be broken) but they may also give hints on how the system could go about finding a solution.

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.

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 [12] or Choco [13]. 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 [14] or PyCSP3 [15].

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 [16].

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 [17, 18].

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 ded-

icated 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 [19], 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 Implementation

After experimenting with other tools of CP we settled on the Choco Solver [13], an open-source Java library for CP. This tool allowed us to mix the versatility of the java language with a suite of known constraints that come implemented with Choco such as the "Sum Constraint" where the sum total of values in an array is constrained to be within a set range of values or to be equal to a value. But crucially Choco allows for custom constraints to be created by defining how the effect of said constraints are propagated, this turns out to be how we create and apply the constraints needed to enforce the 50ha limit of continuously harvested forest area. We opted to base our work on the Choco Java constraint programming library, as we had already developed a dedicated global constraint in the form of a custom propagator. Other constraint programming libraries which we considered using include Gecode [20] and IBM ILOG CP Optimizer [21].

4.1 Model Description

The process of implementing the problem entails firstly setting up 2 data structures, one is an array of `Node` objects called `Nodes` where each Node contains the area of the MU and the IDs of its adjacent MUs. The other array is called `MUS`, it has as many variables as there are MUs in the forest, it is a constraint variable array so each variable does not have a set value but a domain of possible values. These possible values are the possible prescriptions that may be applied in the corresponding MU and, after setting up the constraints, during the solving process the solver will attempt remove values from the domains of these variables as it attempts to find a state where

the every variable has a value assigned to it that does not break the constraints between the variables. Another constraint variable array is called `WoodYields`, each index of the array corresponds to the wood yielded by harvesting/thinning one of the MUs in the forest when applying the prescription that the solver picked for the MU (this process involves using the "Element" constraint from the Choco-Solver framework). This is done so that once a solution is found the contents of this array can be summed up to obtain the total amount of wood yielded by that solution. This is done analogously for the other criteria we want to optimize. Once the setup is done we iterate through the main loop, as shown in Listing 1.

Listing 1 Main Loop

```
for (Var node in Nodes) {
    CreateConstraint (node, MUS, MAXLIMIT);
}
SumOfWood = sumConstraint(WoodYields, range=[−999999, 999999]);
SumOfCrit2 = sumConstraint(Crit2Totals, range=[−999999, 999999]);
```

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 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 continuous 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, with reference to the maximum cut area requirement, as shown in Listing 3.

Listing 2 Custom Propagator

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

Listing 3 Propagator Helper

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

After leaving the main loop, the model has been fully setup.

If the problem is one of constraint satisfaction and not optimization the solver can now be activated and the solution, if found, will be written onto an input file.

The default search strategy that Choco-Solver uses to assign integer values to the constraint variables is based on attributing weights to these values and starting the assignment process with the lowest bound variable so the results are deterministic.

4.2 Multi-criteria Optimization

Multi-Criteria optimization requires the user to inform the solver of which criteria to optimize, and the solver cannot mix and match maximization problems with minimization problems. If the user wants to maximize most variables and minimize others, he must convert the minimization problems into maximization problems by switching the sign of their criteria, as was done in the case of the Vale de Sousa forest with soil loss.

Then, the solver runs on a loop to find as many valid solutions as possible, so solutions that simply do not break the 50ha limit.

Theoretically, it is going to reach every possible valid solution. In practise, the search stops with a memory error or, given that the search may take too long otherwise, when a time limit set by the user is reached.

Once the loop terminates the solver utilizes the objective criteria of all the solutions it found as points to calculate the Pareto efficient points and therefore the Pareto efficient solutions, which it writes to an output file so we can plot the Pareto Frontier and pick a point to paint on an output map.

5 Experimental Evaluation

The testing was done on a laptop running Ubuntu 20.04.3 LTS, with 4GB of available RAM and 4 cores. The code was compiled using the Java SDK version 8.

It should be noted that no solution for the full problem could be found in a reasonable time frame on this platform: the complete problem includes 1406 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.

5.1 Solving the Problem by Sub-region

Paredes and Penafiel

In regards to simple constraint satisfaction problems valid solutions are always found for both of these sub-regions separately and even when solved together.

Regarding multi-criteria optimization for Paredes and Penafiel together the results depend of the number of criterion which we are trying to simultaneously optimize, this type of optimization depends on the solver finding multiple valid solutions. There is a complexity cost in the number of constraints for each criterion the solver has to consider when finding these solutions and while it generally will find valid solutions when optimizing 1 to 4 criteria, depending on the criterion, more than that will leave the solver running for a long time without finding a valid solution.

Being able to find valid solutions for Paredes and Penafiel together is significant since these 2 sub-regions are completely separated from the Paiva sub-region so they can be treated as a separate problem from Paiva.

Paiva

The Paiva sub-region is a source of difficulty for the implementation because the solver is left running for hours and hours and a valid solution is never be found. As this sub-region proved to be the most complex in the forest we deemed it expedient to divide the Paiva sub-region into 3 separate sub-regions which are individually solvable, with the goal of building a Pareto frontier for each of the sub-regions and "joining" the Pareto frontiers for a complete Paiva Pareto frontier. By closely observing the sub-region an attempt was made to divide Paiva in spots where there are few adjacent MUs between sub-regions, ending up with PaivaWest (Pink), PaivaEast (Red) and PaivaIslands (Yellow). PaivaWest is completely separated from PaivaIslands, PaivaEast is only connected to PaivaIslands through 1 MU and PaivaWest is connected to PaivaEast through few MUs. So the forest is divided as shown in Fig. 2.

Because this "joining" of the solutions is being done outside of the Constraint Programming implementation, there is no guarantee that the solutions in this complete Paiva Pareto frontier will not break the 50ha area limit. Therefore we decided to first find a single valid solution to the "Contact Zone" (the group of MUs that are close to the border of 2 adjacent sub-regions) between PaivaWest and PaivaEast, also shown in Fig. 2, and "lock" the pairs of MU/Prescriptions of that solution for the solving process of PaivaEast and West, guaranteeing that the solutions found enforce the 50ha limit between sub-regions. We also developed a script to verify if a solution breaks the 50ha limit.

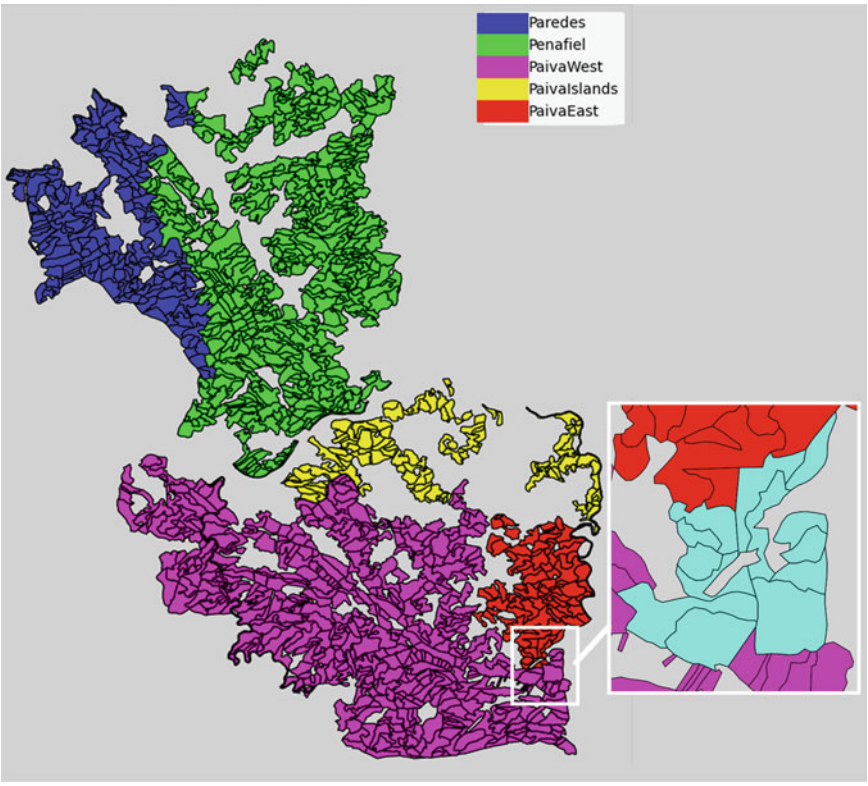

Fig. 2 Vale de Sousa forest divided by sub-region and contact zone

5.2 Joining the Sub-regions

Let's assume that we have 2 sub-regions with N1 and N2 Pareto points respectively. This "joining" process works by "adding" each Pareto efficient point of one sub-region to each Pareto efficient point of the other sub-region, yielding $N1 \times N2$ points. Since this may result in a large number of points some filtration is required to reduce this number. At first we round up the objective values of each point (for example by 4 decimal digits) and then we eliminate the non-efficient points (as well as points with the same objective values). After that we combine the "filtered" points of one sub-region with the "filtered" points of the other sub-region. From the set of obtained points we remove the dominated ones.

Other sub-regions are "joined" with the resulting Pareto set in the same manner successively.

Once the final Pareto frontier is built the user can pick a point on the Decision Map (see Fig. 3), learn the effects of applying the corresponding solution and observe its application throughout the planning horizon.

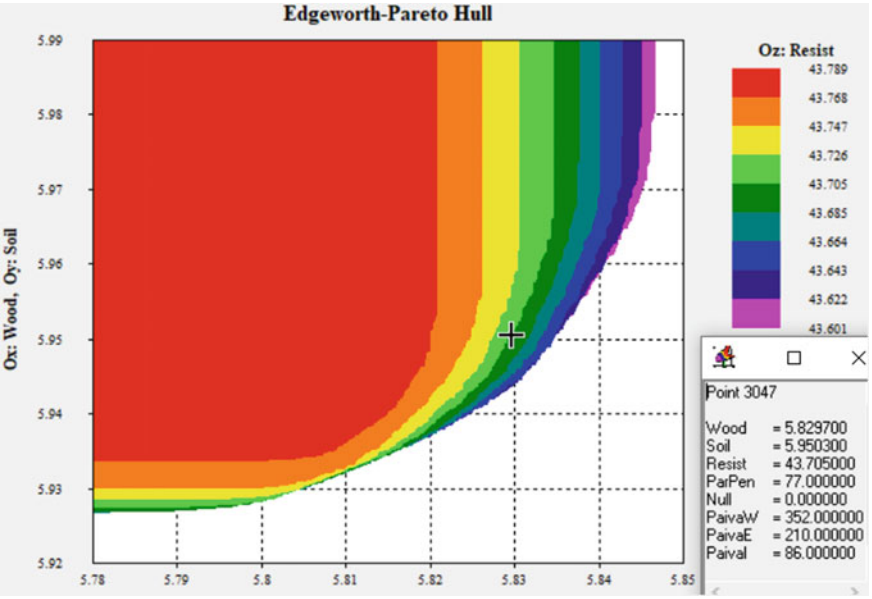

Fig. 3 Visualization of 14833 Pareto points for the complete Vale de Sousa forest

5.3 Results

Firstly valid solutions were found for 3 criteria, wood yield, soil loss and fire resistance, for all sub-regions. The solver was left running until a memory error occurred and all the solutions found until that point were considered, then the Pareto efficient solutions of each sub-region were joined as previously explained. The execution time until a memory error occurs varies depending on the sub-region, as show in Table 1

In our implementation to "join" the results firstly the Pareto points of PaivaWest are joined with the ones from PaivaEast, yielding 103246 possible points but after filtering out the dominated points only 3498 remained. Then once these results are joined with PaivaIslands 9166 points are obtained. Finally, joining these points with

Table 1 Execution details with all sub-regions

Sub-region	Execution time (h)	Valid solutions	Pareto solutions	After filtering
ParPen	04:08	9932	813	226
PaivaIslands	06:00	21555	578	284
PaivaWest	00:46	9364	742	247
PaivaEast	03:40	11534	874	418
CompleteForest	–	–	16901	14833

the 226 filtered points from Paredes+Penafiel yields 16901 points (or complete forest solutions), furthered reduced to 14833 points after filtering.

A point is then picked utilizing a visualizer [22, 23], this visualization is obtained by drawing the dominance cones of each point in the Pareto frontier, as shown in Fig. 3 the vertices of these cones are distributed uniformly so we may say that in this case we obtained well distributed points.

The maximum wood yield represented in Fig. 3 is around 5.845 tons of wood, the minimum soil loss is 5.928 tons of soil lost and the maximum fire resistance index is 43.789.

To showcase the results we opted to use as an example the compromise point marked by a cross in Fig. 3, yielding 5.8297 tons of wood, 5.9503 tons of soil lost and 43.705 fire resistance index.

The point we picked as an example has a corresponding solution which assign a prescription number to each MU. This solution may then be visualized on maps rep-

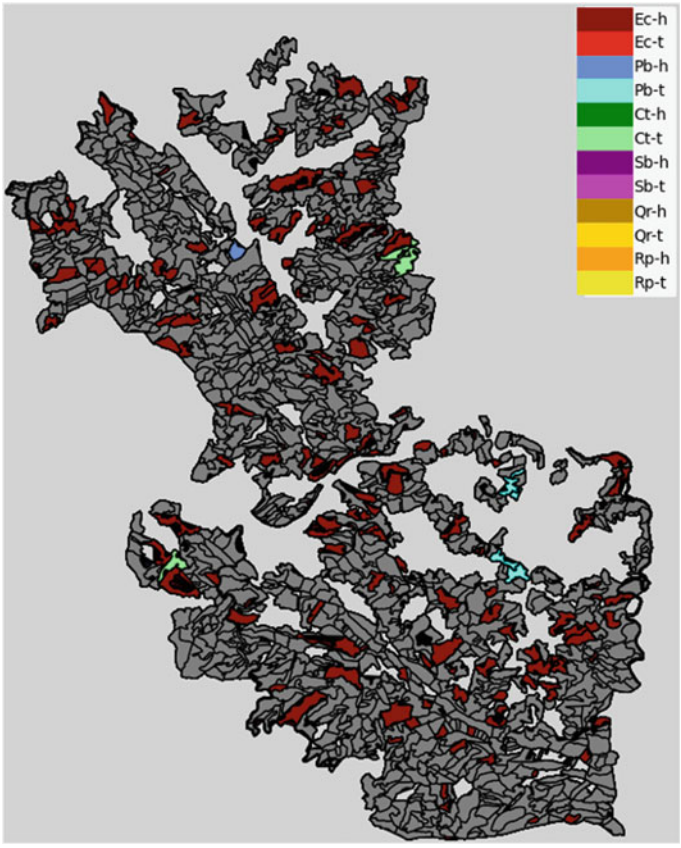

Fig. 4 Example of an output map for a full solution of the Vale de Sousa forest, year 2041

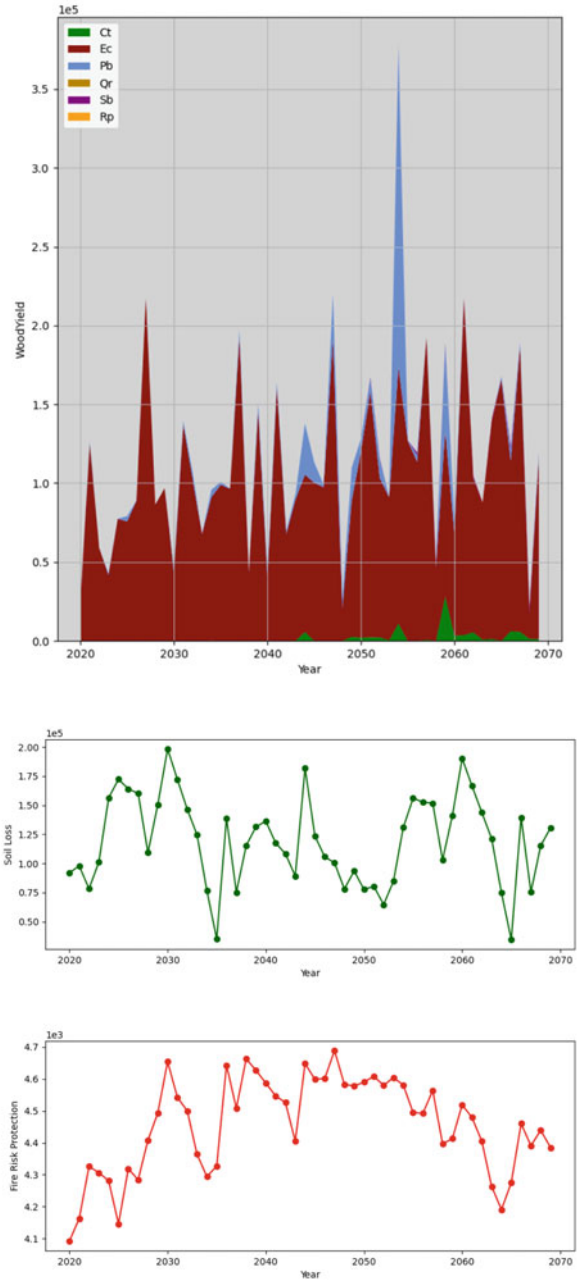

Fig. 5 Graphs showing the year by year values of the wood yield, soil loss and fire risk protection of the chosen solution

resenting management units on which the harvest or thinning operations are applied in each year throughout the 50 year planning horizon.

Figure 4 showcases an output map corresponding to the year 2041 where the colors represent both the species of tree planted in the MUs and the action to be taken upon them, for example dark red represents harvest of eucalyptus (Ec-h) and light green represents thinning of chestnut trees (Ct-t). Figure 5 showcases 3 graphs visualizing the values of the criteria obtained each year.

6 Conclusions and Future Work

In this work we solved the large scale multi-objective Vale de Sousa forest managing problem with adjacency constraints by applying a Constraint Programming approach.

The problem was divided into computationally manageable sub-problems and for each of them a Pareto frontier was calculated by using the Choco-Solver.

These partial Pareto frontiers were combined into a Pareto frontier for the entire Vale de Sousa region. The proposed methodology allowed us to obtain solutions for large scale multi-objective forest scheduling problems in a bounded time frame.

In future developments, we'll study the Vale de Sousa problem with more than 3 criteria being considered and work on improving the constraint programming model and using metaheuristic search procedures.

Acknowledgements The authors acknowledge the Portuguese Fundação para a Ciência e a Tecnologia (FCT) for supporting the development of the system presented in here through the MODFIRE grant (PCIF/MOS/0217/2017).

References

1. Eloy, E., Bushenkov, V., Abreu, S.: Constraint modeling for forest management. In: Tchemisova, T.V., Torres, D.F.M., Plakhov, A.Y. (eds.), Dynamic Control and Optimization, Cham, pp. 185–200. Springer International Publishing (2022)
2. 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)
3. 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)
4. Gunn, E., Richards, E.: Solving the adjacency problem with stand-centered constraints. Canadian J. Forest Res. **35**, 832–842 (2005)
5. Gharbi, C., Ronnqvist, M., Beaudoin, D., Carle, M.-A.: A new mixed-integer programming model for spatial forest planning. Canadian J. Forest Res. **49**, 1493–1503 (2019)
6. Apt, K.: Principles of Constraint Programming. Cambridge University Press (2003)
7. Murray, A.: Spatial restrictions in harvest scheduling. Forest Sci. **45**(1), 45–52 (1999)
8. 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)
9. Pascual, A.: Multi-objective forest planning at tree-level combining mixed integer programming and airborne laser scanning. Forest Ecol. Manag. **483**, 118714 (2021)

10. 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**(9) (2021)
11. Marques, S., Bushenkov, V.A., Lotov, A.V., Marto, M., Borges, J.G.: Bi-Level participatory forest management planning supported by pareto frontier visualization. Forest Sci. **66**, 490–500 (2019)
12. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling. In: Modeling and Programming with Gecode, Christian Schulte and Guido Tack and Mikael Z. Lagerkvist. Corresponds to Gecode 6.2.0 (2019)
13. Prud'homme, C., Fages, J.-G., Lorca, X.: Choco Solver Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2016)
14. 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.), Principles and Practice of Constraint Programming—CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, 23–27 Sept. 2007, Proceedings, vol. 4741 of Lecture Notes in Computer Science, pp. 529–543, Springer, 2007
15. Lecoutre, C., Szczepanski, N.: Pycsp3: Modeling combinatorial constrained problems in python (2020). arXiv:2009.00326
16. Michel, L., Hentenryck, P.V.: Constraint-based local search. In: Martí, R., Pardalos, P.M., Resende, M.G.C. (eds.), Handbook of Heuristics, pp. 223–260. Springer (2018)
17. 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 (2018)
18. Régin, J., Malapert, A.: Parallel constraint programming. In: Hamadi, Y., Sais, L. (eds.) Handbook of Parallel Constraint Reasoning, pp. 337–379. Springer (2018)
19. Boussemart, F., Lecoutre, C., Audemard, G., Piette, C.: Xcsp3-core: a format for representing constraint satisfaction/optimization problems (2020). arXiv:2009.00514
20. Schulte, C., Stuckey, P.J.: Efficient constraint propagation engines. ACM Trans. Program. Lang. Syst. **31**(1), 2:1–2:43 (2008)
21. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP optimizer for scheduling. Constraints **23**, 210–250 (2018). Apr
22. Lotov, A., Kamenev, G., Berezkin, V.: Approximation and visualization of pareto-efficient frontier for nonconvex multiobjective problems. Dokl **66**, 260–262 (2002)
23. Lotov, A., Bushenkov, V., Kamenev, G.: Interactive decision maps: approximation and visualization of pareto frontier. Springer (2004)