Optimization and Algorithms

Project 2

Carlos Costa - Dominik Giynski - Tomas Mendes

Instituto Superior Tcnico

***Abstract*—This paper aims to find the optimal harvesting times for forest's in Portugal, taking into account some efficiency, adjacency and variation constraints to allow for a sustainable planning and management of our forests, which are one of our most precious natural resources.**

I. Introduction

The goal of this project is to optimize forest management by using algorithms to help decide the appropriate time for harvesting a given forest, in such a way that allows for the best long term forest management planning and the highest profit.

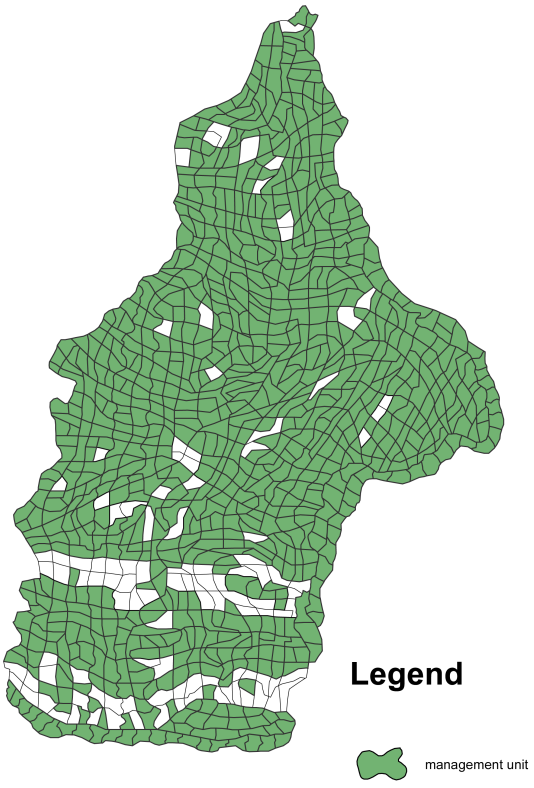

Fig. 1. Example of a forest divided into management units

This project is based on an article published in the European Journal of Operational Research, which tackles these issues for forests in Norway. In that article, the authors divided some Norwegian forests into smaller management units (MUs), and then used data they had compiled about each of those MUs to run the algorithms and constraints they designed in order to find the optimal times for harvesting each MU.

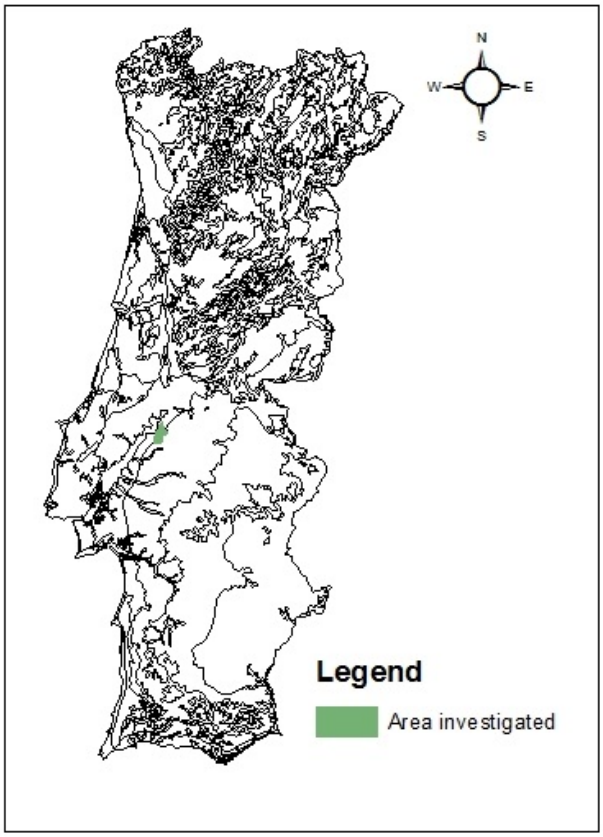

Fig. 2. Location of Chouto Parreiras's forest

In this paper, we'll be applying the conclusions and methods described on that article to a Portuguese forest. This forest, called Chouto Parreira, is composed by 889 stands (which we'll be calling MUs) and covers a total of 12435 ha. Figure 1 shows the rough shape of this forest as well as the 889 MUs, and Figure 2 shows the forest's location within the Portuguese map.

Chouco Parreiras' flora is predominantly dominated by

eucalyptus. Known to some as a plague, do to their incredible resilience and reproduction speed, eucalyptus are also the cornerstone of Portuguese pulp and paper manufacturers, such as Celbi and Soporcel, due to providing the raw materials necessary for the making of these products, which account for a significant portion of Portugal's exhortations. Because of this, maintaining and harvesting these forests responsibly and in an efficient way becomes a priority for these companies, and makes the type of research contained in this paper essential for the long term health of some of Portugal's businesses.

II. Problem definition

In this paper we'll be using the same cost function and a lot of the same constraints that the authors of the article used. The cost function focuses on maximizing the profit made from harvesting these MUs, and is defined as:

$$\max \quad Z = \sum_{i=1}^{N}\sum_{j=1}^{M_i} NPV_{ij} \cdot x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{M_i} x_{ij} = 1, \quad \forall i \in N \tag{2}$$

$$VH_t = \sum_{i=1}^{N}\sum_{j=1}^{M_i} vh_{ijt} \cdot x_{ij}, \quad \forall t \in T \tag{3}$$

$$0.9 \cdot VH_{t-1} \leqslant VH_t \leqslant 1.1 \cdot VH_{t-1}, \quad t = 2, \ldots, T \tag{4}$$

$$x_{ij} + x_{i'j'} \leqslant 1, \qquad \forall (i,i') \in I, \quad \forall (j,j') \in J \tag{5}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in N, \quad \forall j \in M_i \tag{6}$$

NPV stands for Net Present value, and we wish to maximize our cost function Z. This function however is subject to a lot of constraints, which are mostly to guarantee the efficiency of the harvesting. Equations (3) and (4) are what we'll call in this paper our variation constraint, which says that the volume harvested in each year has to be within a 10% variation of the volume harvested in the previous year. Equations (2) and (6), are there to guarantee that in the time period considered, which is 15 years, no MU is harvested more than once. Equation (5) is what is called an adjacency constraint, and we will be ignoring it during the first half of this paper since it adds a lot of complexity to this problem. Finally, we introduce a final age constraint, specific to this problem. This age constraint's purpose is to prevent some MUs that are to young from being harvested, and is defined as:

$$x_{ij} = 0, \forall ZerosVar_{ij} = 1 \tag{1}$$

Where ZerosVar is a 889x15 which contains the information of which and when there are forests to young to harvest.

III. Solving the Problem with Matlab

To solve this problem in Matlab, we first need to have the variables correctly defined and in the right units. In our data, we were given the variables T=15 and N=889, which define respectively the number of years and management units that we'll be considering. We were also given the variables NPV and Vol, which define the Net Present Value and the volume of each MU per year, but are both defined in squared meters, which is not very useful for our purposes. The first step of the project then, was to take these two variables and multiply them by the diagonal of the vector Area, a 889x1 vector which defines the area of each MU. Within the original data we also had the variable NPV, a 889x15 matrix which defines the Net present value, and the variable adj, which is used to compute the adjacency constraint.

A. Using CVX

With this out of the way, we first try to solve this problem using CVX, which is a modeling system for convex optimization. This model functions in a very straightforward way, so inputting our cost functions and constraints is fairly simple, requiring only a little bit of Matlab knowledge. The program takes between 9 and 10 seconds to solve, but it does so with a good accuracy. After analyzing the result, and rounding every answer to the fourth decimal place (we did this mostly to facilitate comparisons later), we concluded that CVX was able to fulfill the conditions of every constraint. This, however is only possible since CVX forces the relaxation of one of our main constraints: To guarantee that we only harvest a MU once in the time period of 15 years, the optimal output of our function should be a matrix in which each row should contain a single one and fourteen zeroes. This type of binary programming however, is not possible in CVX, so the values it outputs can be any continuous number between zero and one. After rounding, we can see that this isn't an issue for most of the MUs, since it gives binary outputs for the majority of them, but in the cases where this doesn't happen, further decisions need to be made in order to have a working model.

Despite all of this, the output of CVX was able fulfill the volume harvested variation constraint every time. For this inequality, we relaxed slightly the original parameters, from 10% maximum variation allowed to 20%. This decision was made mainly to relax some of the computational burden. Figure 3 shows the projected variation of volume harvested per year using the CVX Model. Notice that The volume constantly increases, but it never increases by more than 20% each year. All code related to CVX can be found in the file "CVX-Project.m".

B. Using Linprog

Linprog is a Matlab function that solves optimization problems through linear programming. Solving our problem with this function was a lot trickier, since it's inputs are very different from CVX's. For starers, this function can only output a vector, so instead of dealing with the 889x15 matrices that we were used to, we now needed to start using 1x13335 vectors. The function also can only minimize a problem, and since we wanted to maximize our profits, our cost function had to be negative, to turn the original problem into a minimization one.

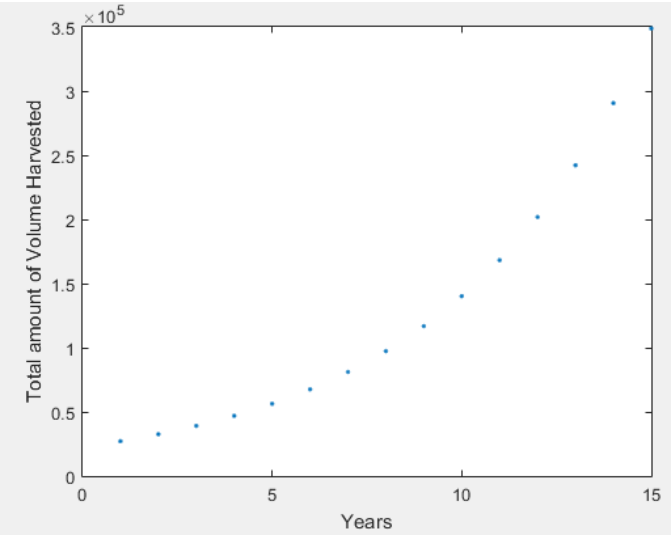

Fig. 3. Example of a forest divided into management units

In this function, we define inequality constraints by inputting a matrix A and a vector B, such that A*X should be less or equal than B. Each row of matrix A corresponds to one inequality of equation (4), so, to be able to define this inequality we needed our matrix A to have 28 rows, each with 13335 columns, and our Vector B to have 28 rows. Since half of these inequalities are of the "greater or equal" type, instead of the "less or equal" that linprog assumes a default, we had to multiply the corresponding rows on matrix A by -1.

Equality constrains are defined in a very similar way, requiring a matrix Aeq and a vector Beq such that Aeq*X=Beq. Here, we'll need our matrix Aeq to have the size 889*13335 to satisfy that the sum of every MU equals one. To handle the age constraint we add an extra row at the end of this matrix, and guarantee that the sum of every MU that's too young to be harvested is zero. Since the output for each MU can only vary from 0 to one, this guarantees that the constraint will be met. The code for this part is in the file "linprog-project.m".

This method proved to be less accurate than CVX, since, despite producing a solution in a tenth of the time, the solution found violated the variation constraint 6 out of 28 possible times. These violations were never very severe though, since the solution found was always within a 22% variation each year.

C. Comparing CVX with Linprog and analyzing the outputs

The Linprog program runs considerably faster then CVX, taking on average less than a second. After rounding to the fourth decimal place, and comparing the output to CVX's, we see that they match in 13329 out of 13335 cells. These 6 differences however, are responsible for linprog's less accurate solution, so it's fair to say that linprog sacrifices a bit of accuracy for speed. This difference is incredibly slight, though, and when plotting both Cvx and Linprog in the same graph as we did in figure 3, the two plots overlap completely and it's impossible to make out the differences.

Since both of the outputs for these two methods are continuous values and not pure binary numbers as we wanted, we implemented two methods to smooth out these outputs. First, we use a selection method, where, for each MU, we select the highest number among every cell, replace it with one set all other numbers in that row to zero. After applying this method, to CVX and Linprogs' output, and comparing them with each other, we found that they now matched in 100% of points. This new solution however, even though it comes in the binary form that we wanted it to come in, fails at satisfying the variation constraints 7 out of 28 times, so it's less accurate than linprog. This, however, is to be expected. When rounding out the values in this manner we won't get a perfect solution, otherwise these methods would have found it in the first place. We found this solution to be a reasonable one, because despite not rigorously abiding by every rule we set, it deviates from those initial rules in an acceptable manner.

The Monte Carlo method was also used to try and smooth out the outputs from Cvx and Linprog. Monte Carlo's method is probability based. The way it works is that, in each row of our solution where the ouputs are not binary, we go cell by cell and replace their numbers with the sum of all of the previous numbers, eventually leading up to one. Then we use Matlabs' rand function to generate a number between zero and one and use this number to select the first cell of the row which contains a number greater than the one which was generated. This method is, obviously, random, so running it multiple time will give different results. Despite this, by sheer probability, it can some times reach a very reasonable solution, and, from our simulations, it's able to find a solution where it only violates the variation constraint 6 out of 28 times, which is a better result than the previous method that we explored. Figure 4 shows five iterations of this method in a graph, to help better get a sense of how this function can variate.

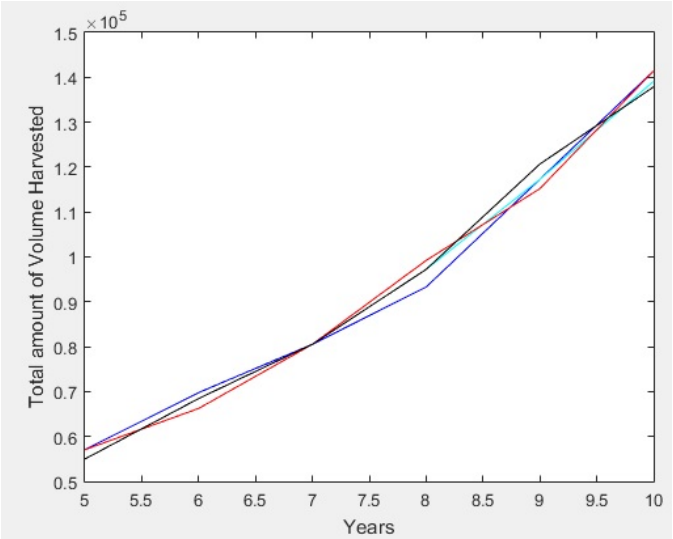

Fig. 4. Plot of several iterations of Monte Carlo

D. Using intlinprog

The next step in our project was to use the matlab function intlinprog to solve the same problem. Intlinprog is a mixed-integer linear programming solver, which functions in a fairly similar way to linprog, but only produces integer outputs. This means that, by setting the allowed outputs of the function to zero and one, we can theoretically get an optimal solution to our problem. This however, in practice, is not the case. Binary programming on this scale is incredibly complex, and when running this function with the original parameters, that is 889 MUs and a time period of 15 years, the program can run on for hours with no end in sight.

In an attempt to get the function to work, we tried significantly reducing the scope of the problem. First, we tried to limit the problem only to the first 300 MUs, but the problem proved to be still too complex, since the function still took a really long time running. We experimented with a further reduction to 100 MUs, but we then realized that we couldn't just pick any set of 100 MUs, because some factors like the ZerosVar variable could make the problem unfeasible. We then tried to handpick smaller sets of MUs to try and get some results from this function. After some trial and error, we got some success. With a set of just 15 MUs (the bare minimum number of MUs the function can accept due to the variation constraint), we got a feasible solution out of intlinprog, but it still took nearly 40 minutes to run. Figure 5 illustrates this solution. By inspecting the image, it's clear that the failure rate is high (9 out of 28), but it's also not a fair comparison since the input data is too small to have an optimal solution. Due to the limitations of the method, it's hard to find an interval of MUs where the method can function correctly.

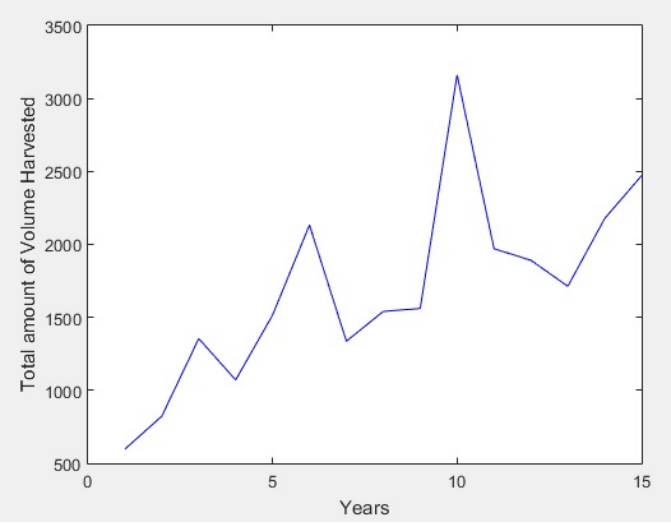

Fig. 5. Plot of Volume harvested per year using linprog

These extremely long running times are in part due to the high precision that intlinprog has set as a default, but the main reason is still the very demanding nature of binary programming, which makes methods like CVX and Linprog, in which we relax our constraints, very attractive alternatives to this method, despite all of their drawbacks. The code for this part of the roject can be found on the fie "intlinprog-project".

IV. Using Adjacency Constraints

In the second phase of the project, limitations on the adjacent stands were introduced into the model. As according to the guidelines, neighboring stands are not allowed to be harvested in the same time period. The data on the stands sharing an edge are stored in a matrix of a 1803x2 size. Adjacency constraint is of the form of the equation (5).

Thus, for each pair of adjacent stands (1803 in total), and for each period of time, these conditions were added to the CVX model, which causes an increase in the number of restrictions up to 1803 * 15. It significantly increases the time necessary to find a solution to the problem. In the case of forest planning with adjacency constraints, cvx solver needs about 1000s to solve the problem. Comparing this with the execution time for the model without adjacency constraints, which takes less than a second, it can be seen just how much this new constraint complicates the model. This already massive increase in programming time would be even grater if we were using binary methods, which makes continuous methods like CVX the only practical way of solving this problem. We can interpret the continuous variables produced by Cvx by using, for example, the Monte Carlo method, which helps us turn the output of the CVX into binary variables. Doing this, however, does not guarantee that the original constraints are met, but it can still produce an acceptable solution.

We compute failure rate for our model with this restriction on the number of occurrences of solution equal to approx. 0.5. This ratio for the various allowed variation in the flow of the harvested volume between periods is approximately 1% of the total output. The code for this part of the project can be found in the file "Cvx-adj-project".

V. Conclusion

During the writing of this paper, we learned just how important it is to accept certain reasonable limitations on your initial model in order to be able to efficiently generate a reasonable solution. It's clear that solving this problem using binary methods only (intlinprog) would be completely unfeasible, specially if we tried to add the adjacency constraint. A much better approach is to relax our variables slightly and then try to interpret that result and transform it into a solution that best fits our model. In the specific case of this paper, we concluded that the best approach would be to use the CVX model, and then run several iterations of the Monte Carlo's method to find the solution that best fits all of our constraints. This way, we can get a good approximation of an optimal solution while keeping the computational burden relatively small.

References

[1] Paulo Borges , Tron Eid, Even Bergseng, "Applying simulated annealing using different methods for the neighborhood search in forest planning problems" *European Journal of Operational Research.*, 2014.