UNIVERSITY OF LISBON
INSTITUTO SUPERIOR TÉCNICO
MSC. COMPUTER SCIENCE AND ENGINEERING

TÉCNICO LISBOA

Algorithms for Computational Logic

Project 2 - Report

[SMT] Harvesting Scheduling

90775 - Ricardo Fernandes

LISBON, NOVEMBER 2022

1 Problem

The problem for this project can be summarized with units of farming land which have different profits depending on the period which are harvested. There must also be natural reserve units which are units that cannot be harvested and must be contiguous. The area for these must be greater or equal than a supplied minimum area. So, each unit as a area associated with.

1.1 Definitions

For this report the following definitions will be used.

$$U = \{1, ..., n\}, \qquad T = \{1, ..., k\}, \tag{1}$$

$$A = [A_1, A_2, ..., A_n] : \text{Areas for each unit} \tag{2}$$

$$P_{ij} : \text{returns the profit of the unit } i \text{ at period } j \tag{3}$$

$$H_{ij} : \textit{True} \text{ if unit } i \text{ was harvested at period } j \tag{4}$$

$$\text{NR}_i : \textit{True} \text{ if unit } i \text{ is a natural reserve} \tag{5}$$

$$D_i : \text{depth of unit } i \tag{6}$$

2 Encoding

The encoding of this problem is pretty similar to the first one, where we define the periods of harvest for each unit as a Boolean, meaning that for example, if unit i was to be harvested at period j, a Boolean variable representing this would be True.

$$\forall_{i \in U} \forall_{j \in T} \text{Bool}(H_{ij})$$

This time, because SMT and Z3 makes variable usage easier, the natural reserve variables are splat making it easier to understand. NR_i is true if unit i is a natural reserve.

$$\forall_{i \in U} \text{Bool}(\text{NR}_i)$$

To help define natural reserve contiguity, similar to the first part of the project, auxiliary variables are needed. With the help of SMT, we can define the depth of the unit in the tree as a integer variable.

$$\forall_{i \in U} \text{Int}(D_i)$$

3 Constraints

$$\sum_{j \in T} H_{ij} <= 1 \qquad \forall_{i \in U} \tag{1}$$

$$H_{ij} \rightarrow \neg H_{i_n j} \qquad i_n > i, \ \ \forall_{i \in U} \forall_{i_n \in i_{neighbours}} \forall_{j \in T} \tag{2}$$

$$NR_i \rightarrow \bigwedge_{j \in T} \neg H_{ij} \qquad \forall_{i \in U} \tag{3}$$

$$D_i \geq 1 \Leftrightarrow NR_i \qquad \forall_{i \in U} \tag{4.1}$$

$$D_i > 1 \rightarrow (\sum_{i_n \in i_{\text{neighbours}}} D_i = D_{i_n} + 1) = 1 \qquad \forall_{i \in U} \tag{4.2}$$

$$(\sum_{i \in U} D_i = 1) = 1 \tag{4.3}$$

$$\sum_{i \in U} A_i \cdot NR_i \geq A_{min} \tag{5}$$

$$maximize \ \sum_{i \in U} \sum_{j \in T} H_{ij} \cdot P_{ij} \tag{6}$$

1. **A unit of land can only be harvested once.** The sum of true values in the harvesting variables for each unit must be at most one.

2. **Neighbour units cannot be harvested in the same time period.** If unit i is harvested at the period j, then every neighbour of i, i_n cannot be harvested at that same period. Check optimization 5.2.1.

3. **A natural reserve unit must not be harvested.** If a unit is a natural reserve, no harvesting variables for that same unit can be true.

4. **The natural reserve area is contiguous.**

   As mentioned before, singular contiguity can only be defined with the notion of depth of the contiguity, this is accomplished with the help of auxiliary variables.

   We can think of it as a uni-directional tree, where the root is the start of the contiguity. With the following constraints, it's possible to solve the contiguity problem.

   In this implementation, because we use integer variables to define the depth in the tree, we consider that a unit belongs to tree if its depth in greater or equal than one.

   4.1 **A natural reserve unit must be in the tree and a unit that is in the tree is a natural reserve unit.** If unit i is a natural reserve (period $k + 1$) then, then sum of the presences of i in the different depths of the tree must be at least 1. And also the reverse, meaning, if i at least is present in one depth of tree then, it must be a natural reserve.

   4.2 **A unit in the tree has to have one, and can only have one, predecessor.** If unit i is present in the tree at depth d, not being the root level 1, then, the sum of presences of i neighbours, i_n, in the tree at depth $d - 1$ must be exactly 1.

   4.3 **There can only be one root.** The sum of units in tree at depth 1 must be at most 1.

5. **The natural reserve area must be $\geq A_{min} \geq 0$.** The sum of areas of each natural reserve unit (if NR_i is true) must be greater than the established minimum natural reserve area.

6. **Maximize the sum of profits when the units were harvested.**

4 Implementation

The implementation was done with the programming language python, version 3.10, with the library `z3-solver>=4.11.2.0`

For the harvesting H_{ij} and natural reserve NR_i variables a SAT approach (Bool) was chosen after few iterations of optimizations. For the units' depth in three a more SMT approach was implemented, meaning that integers were used.

5 Optimizations

5.1 Approach optimizations

The following optimizations were made after comparing the results of several iterations of the solution.

A The first solution was implemented using only integers for all the same variables mentioned in the solution. This was giving a ridiculously amount of CPU time

B What I though after, was to try to make use of booleans in the natural reserve variables, which led to the NR_i variables. This, increased the performance more than double.

C With the usage of booleans variables increasing the performance, it made me think I used more boolean variables instead of integers, if the performance would increase more. This led to the H_{ij} boolean variables, which increased dramatically the performance.

D With the units' depth in the tree variables left to try to convert to boolean variables, it would not be wise to not do it. So, after implementing a similar to the first part of the project logic (regarding the tree), mixed results were obtained, where, in general, meaning that the sum of the CPU time of the total tests, was more performant than the C implementation, there were cases were this was not true. So, for the sake of the project, for not using the same logic as the first part and use a more SMT logic, I decided to keep the mixed SMT-SAT logic (approach C) implementation.

The following table represents the times for tests:

T1: `t_10_10_10_40_3_10_20.hsp`

T2: `t_10_10_10_40_3_10_30.hsp`

T3: `t_10_10_10_40_3_10_40.hsp`

T4: `t_10_10_10_40_4_8_40.hsp`

CPU time (seg)	A	B	**C**	D
T1	1.990	2.439	**0.959**	0.495
T2	4.556	3.072	**1.234**	2.442
T3	177.434	94.772	**2.273**	63.014
T4	1095.792	367.804	**16.998**	55.845

In conclusion, the **approach C** was implemented, and is the one described in this report.

5.2 *Out-of-the-box* optimizations

The following optimizations are optimizations where decided the solution approach, are made to reduce iteration and redundant/unnecessary constraints.

1. In constraint (2) instead of going through every neighbour of i, to avoid redundant constraints, it's only needed to go through neighbours whose id is greater than i. For example, let's say U_3 is neighbour of U_2, in the iteration where $i = 2$ it will add the clause $H_{2j} \rightarrow \neg H_{3j} \Leftrightarrow \neg H_{2j} \vee \neg H_{3j}, \forall_{j \in T}$; and in the iteration $i = 3$, if there the optimization didn't exist, it would add also the clause $H_{3j} \rightarrow \neg H_{2j} \Leftrightarrow \neg H_{3j} \vee \neg H_{2j}, \forall_{j \in T}$. As we can see, these two clauses are the same.

2. If the natural reserve minimum area is zero, the (3), (4.1), (4.2), (4.3) and (5) constraints are not added.

6 Run

To run the project, install the dependencies: `pip3 install -r requirements.txt`, and then run the main python file: `python3 main.py` or just run `./proj2`

References

[1] Tiago Almeida and Vasco Manquinho. Constraint-based electoral districting using a new compactness measure: An application to portugal. *Computers & Operations Research*, 146:105892, 2022.