Workforce timetabling optimization

- III Iberian Modelling Week -

April 11^{th} - 15^{th}, 2016

Arti Bandhana - FCUP
Marina Pereira - FCUP
Sara Cerqueira - FCUP
Duarte Ribeiro - UNL
Ricardo Silva - UNL
Tiago Nunes - UNL

Coordinator
Prof. Paula Amaral
Assistant Professor at Department of Mathematics FCT-UNL
Member of CMA/FCT-UNL

Contents

1 Introduction

Optimization is based on finding the best solution to a given problem by simplifying and expressing it in mathematical notation in order to create a model describing the problem. This is done by employing different optimization algorithms to the mathematical description of the problem, which consists of an **objective function** $f(\mathbf{x})$, that is to be minimized (or maximized), and a set of **constraints**.

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in X \end{array}$$

Where $f(\mathbf{x}) : X \longrightarrow \mathbb{R}$, with $X \subseteq \mathbb{R}^n$, $n \in \mathbb{N}$, the feasible set determined by the problem constraints.
The objective function expresses the goal of a given problem in terms of variables, $\mathbf{x}$. The objective function, together with the set of constraints, is what builds the mathematical model for the given problem.
Even though the goal is to find the best solution for each problem, some problems are very large and thus finding it can be very time consuming. However, with increasing technical resources, many problems today are manageable. There are many areas where optimization is frequently used, for example finding a suitable timetabling of courses at universities or trying to find the most efficient route for deliveries to certain destinations.

1.1 Linear programming

In linear programming both the objective function and the constraints are linear. A general linear programming problem is a problem of the form

$$\begin{array}{ll} \min & f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

Here, $\mathbf{c} \in \mathbb{R}^n$ is the weight coefficient of the objective function and $A \in M_{m,n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^m$ are the coefficients of the constraints.

The LP model has been applied in a large number of areas including military applications, transportation and distribution, scheduling, production and inventory management, telecommunication, agriculture and more. Linear Programming is broad enough to encompass many interesting and important applications, yet specific enough to be tractable even if the number of variables is large.

1.2 Integer programming

In many real world problems some variables may not take any value. For example if the question is whether a specific project should be carried through or not we would not get any useful information with a linear model. We would need a variable that takes the value one if the project should be realized and zero otherwise. Then we get an IP problem: when all variables may only take integer values. Solving it as an LP problem and rounding the LP solution does not necassarily give the optimal solution. If the variable in the above example can take any value between zero and one, we can not conclude that the project should be realized even if we get 0.99 as the optimal solution. There could be several other aspects that make this choice disadvantageous, for instance if there is not enough money available to complete the project and it is of no use when it is incomplete.

A general integer programming problem is a problem of the form

$$\begin{array}{ll} \min & f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \\ & \mathbf{x} \text{ integer} \end{array}$$

Because of the fact that the optimal solution of integer programming problems can be anywhere in the discrete set of possible solutions, not only in the vertixes of the subjacent feasible set, this kind or problems are much harder to solve than LP ones. The most naive approach to solve an IP problem would be compute all possible solutions and simply choose the best one, but in practice it takes too much time for large problems.
Several techniques have been developed in order to help us solve this large problems, which can be exact approaches or heuristics (non-optimizing methods). One of this methods, which is exact, is the Branch and Bound. The ideia behind this method is to divide iteratively the set of feasible solutions, creating and solving sub problems using linear relaxations, and take advantage of bounds updated so far to indentify the interesting path towards the solution.

1.3 Problem Description

Many companies have to deal with staff timetabling issues. Assigning laboring periods of time to individuals is a complex problem. Having understaffed or overstaffed during work hours seems to be a perpetual problem. How should employees be scheduled to meet the demand of workforce with minimum labor costs satisfying labour rules? Scheduling is a classic operations research problem. Constraints such as shift lengths, employee weekly hours, and minimum number of shifts are often considered. If the company operates in a 24hours base the problem is even more complex since the beginning and end of the shifts must be decided and the sequence of shifts must meet health regulations.
A classic example of this problem is nurse rostering, but many other applications are easy to find like in TAP, EDP, Security enterprises, etc.

1.4 Purpose

The purpose of this report is to, through the use of integer programming, determine a valid timetable for the given problems. The timetable must fulfill all requirements necessary to be practically possible and it should be optimized in order to minimize the cost of not satisfying the work demands. The formulation should be as general as possible and we will apply it to a real TAP problem, described in the Computational Results chapter.

1.5 Limitations

We were forced to work with a limitation which assumes that every worker is available at all times. This means that there will be no consideration taken to the availability of the worker on a certain period. This is the actual way the timetable is done nowadays so this limitation won't be a practical problem.

2 Model M_0 - Just to start

In order to reach to the final model, step-by-step approach was executed whereby in each step the level of difficulty was increased which led to the model being more practical and applicable to real life scenarios. This model was formulated just for the purpose of familiarity with the problem statement, which can give an indication as to what approach needs to be taken into consideration in order to reach to the final model with all constraints considered. As per the problem description, the goal here is to assign teams to different slots minimizing the assignment costs and ensure that every slot has at least one team working.

2.1 Formulation

Given s slots, t teams and the scheduling costs of assigning a team i to a slot j, $c_{i,j}$, we choosed to use the following variables.

$$x_{ij} = \begin{cases} 1, & \text{if team } i \text{ is assigned to slot } j \\ 0, & \text{other wise} \end{cases} \quad , \quad i \in \{1, \ldots, t\}, j \in \{1, \ldots, s\}$$

Since the goal was minimize the cost of assigning each team to a slot we had to take into account our variables and the the costs of assignment:

$$\min \sum_{i=1}^{t} \sum_{j=1}^{s} c_{ij} x_{ij}$$

The constraints that translate the desire to assign at least one team to each slot can be expressed as:

$$\sum_{i=1}^{t} x_{ij} \geq 1, \quad \forall j \in \{1, \ldots, s\}$$

Model M_0 can then be formulated as

$$\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{t} \sum_{j=1}^{s} c_{ij} x_{ij} \\ \text{Subject to:} & \displaystyle\sum_{i=1}^{t} x_{ij} \geq 1, \quad \forall j \in \{1, ..., s\} \\ & x_{ij} \in \{0, 1\}, \quad i \in \{1, ..., t\}, \quad j \in \{1, ..., s\} \end{array}$$

3 Model M_1 - Balance the use of workforce

The next step is to think about the number of slots that each team should work so that there aren't teams working much more than others. In order to balance the use of workforce and given lower and upper bounds, **Lb** and **Ub**, we have to add constraints to the previous model to ensure that each team will work a number of slots between Lb and Ub.

3.1 Formulation

Our variables are the same as before:

$$x_{ij} = \begin{cases} 1, & \text{if team } i \text{ is assigned to slot } j \\ 0, & \text{other wise} \end{cases}, \quad i \in \{1, \ldots, t\}, j \in \{1, \ldots, s\}$$

And given costs c_{ij}, $i \in \{1, \ldots, t\}$, $j \in \{1, \ldots, s\}$, we still want to minimize the assignment costs, such that the objective function remains the same:

$$\min \sum_{i=1}^{t} \sum_{j=1}^{s} c_{ij} x_{ij}$$

Although we are adding new rules to the problem, the constraints from the previous model, related to assign at least one team to each slot, stay unalterable:

$$\sum_{i=1}^{t} x_{ij} \geq 1, \quad \forall j \in \{1, \ldots, s\}$$

The new constraints that translate the desire to keep the number of slots that each team will work between given lower and upper bounds can be written as shown next:

$$Lb \leq \sum_{j=1}^{s} x_{ij} \leq Ub, \quad \forall i \in \{1, \ldots, t\}$$

Therefore, we have a new ILP model that guarantees each slot will have at least one team assigned and each team will work between a limited number of slots, minimizing the assignment costs, that can me formulated as below.

$$\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{t} \sum_{j=1}^{s} c_{ij} x_{ij} \\ \text{Subject to:} & \displaystyle\sum_{i=1}^{t} x_{ij} \geq 1, \quad \forall j \in \{1, ..., s\} \\ & \displaystyle Lb \leq \sum_{j=1}^{s} x_{ij} \leq Ub, \quad \forall i \in \{1, ..., t\} \\ & x_{ij} \in \{0, 1\}, \quad i \in \{1, ..., t\}, \quad j \in \{1, ..., s\} \end{array}$$

4 Model M_2 - Working with demands

Taking another step towards our final model, the next approach includes the previous restrictions that at least one team should be assigned per slot and that each team will work in a number of slots between Lb and Ub, but now the goal is to minimize the cost of not satisfying the demand. For every slot is given a cost for having less teams than necessary and, because of this, we have to add new variables and modify our previous objective function in order to model this problem.

4.1 Formulation

Given the demand for each slot d_j, $j \in \{1, \ldots, s\}$, the costs of failing the demand c_j, $j \in \{1, \ldots, s\}$, the variables for our new model are:

$$x_{ij} = \begin{cases} 1, & \text{if team } i \text{ is assigned to slot } j \\ 0, & \text{other wise} \end{cases}, \quad i \in \{1, \ldots, t\}, j \in \{1, \ldots, s\}$$

$$v_j = \text{difference between the demand and the service at slot } j, \quad j \in \{1, \ldots, s\}$$

Since we want to minimize the cost of failing the demand, we had to change the objective function to:

$$\min \sum_{i=1}^{s} c_j v_j$$

The role of the new variables introduced will be explained later.
All previous constrains stay unalterable and for the purpose of only taking in account failing the demand, not by having more teams working than necessary, but for having less, new constrains were needed:

$$v_j \geq d_j - \sum_{i=1}^{t} x_{ij}, \quad \forall j \in \{1, ..., s\}$$

$$v_j \geq 0, \quad \forall j \in \{1, ..., s\}$$

Note that if $d_j - \sum_{i=1}^{t} x_{ij} \leq 0$, then the fact that we want to minimize the objective function and the constrain $v_j \geq 0$ forces that $v_j = 0$. And if $d_j - \sum_{i=1}^{t} x_{ij} > 0$, again due to the fact that we are minimizing the objective function, mandatorily $v_j = d_j - \sum_{i=1}^{t} x_{ij}$. These are the constraints that guarantee that only the costs of not meeting the demand by default are considered, since if the service exceeds the demand, the portion added to the cost function is 0.

$$\begin{array}{ll} \min & \sum_{j=1}^{s} c_j v_j \\ \text{Subject to:} & \sum_{i=1}^{t} x_{ij} \geq 1, \quad \forall j \in \{1, ..., s\} \\ & Lb \leq \sum_{j=1}^{s} x_{ij} \leq Ub, \quad \forall i \in \{1, ..., t\} \\ & x_{ij} \in \{0, 1\}, \quad i \in \{1, ..., t\}, \quad j \in \{1, ..., s\} \\ & v_j \geq d_j - \sum_{i=1}^{t} x_{ij}, \quad \forall j \in \{1, ..., s\} \\ & v_j \geq 0, \quad \forall j \in \{1, ..., s\} \end{array}$$

5 Model M_3 - Satisfying the labour rules

The final step of our problem is to extend the previous model to satisfy certain labour rules, accomplishing the established final goal.

The considered labour rules are:

1. Teams are assigned to sequences of ϵ slots;
2. After each assignment, teams must rest at least δ consecutive slots.

5.1 Formulation

Once again, we consider the sets of variables:

$$x_{ij} = \begin{cases} 1, & \text{if team } i \text{ is assigned to slot } j \\ 0, & \text{other wise} \end{cases} , i \in \{1, \ldots, t\}, j \in \{1, \ldots, s\}$$

$$v_j = \text{difference between the demand and and the service at slot } j,\ j \in \{1, \ldots, s\}$$

However, we need to introduce additional ones:

$$B_{ij} = \begin{cases} 1, & \text{if team } i \text{ begins a sequence of slots at slot } j \\ 0, & \text{other wise} \end{cases} , i \in \{1, \ldots, t\}, j \in \{1, \ldots, s\}$$

$$E_{ij} = \begin{cases} 1, & \text{if team } i \text{ finishes a sequence of slots at slot } j \\ 0, & \text{other wise} \end{cases} , i \in \{1, \ldots, t\}, j \in \{1, \ldots, s\}$$

Our objective function remains the same as before, as we still want to minimize the gap between the demand and service for each slot:

$$\min \sum_{j=1}^{s} c_j v_j$$

All the constraints of the previous model are aswell considered, otherwise this would not be an improvement of that problem.

We will introduce new constraints in order to satisfy the referred labour rules, altough we start with the ones related to the new variables formulation.

The new variables are binary, so:

- $B_{ij} \in \{0, 1\}, i \in \{1, ..., t\}, j \in \{1, ..., s\}$
- $E_{ij} \in \{0, 1\}, i \in \{1, ..., t\}, j \in \{1, ..., s\}$

Consider now the following set of constraints:

$$x_{i(j+1)} - x_{ij} = B_{i(j+1)} - E_{ij}, \forall i \in \{1, ..., t\}, \forall j \in \{1, ..., s-1\} \tag{1}$$

If team i starts a sequence of slots at *j+1*, then i must not be assigned to slot j, $x_{ij} = 0$, and must be assigned to slot *j+1*, $x_{i(j+1)} = 1$. In this case, the first member of equation (1) is 1, implying that $B_{i(j+1)} = 1$, as intended. Similarly, if team i finishes a sequence of slots at j, then necessarily $x_{ij} = 1$ and $x_{i(j+1)} = 0$, implying by (1) that $E_{ij} = 1$.

In all the other cases, that is, if $x_{i(j+1)} = x_{ij}$, we want to ensure that $B_{i(j+1)} = E_{ij} = 0$, but looking at (1), both can be 1 aswell. To overcome this situation, we introduce the next set of constraints:

$$B_{i(j+1)} + E_{ij} \leq 1, \forall i \in \{1, ..., t\}, \forall j \in \{1, ..., s-1\}$$

Finally, regarding the new variables good functioning, (1) does not treat B_{i1} and $E_{is}, \forall i$, so we add a new set of constraints.

- $B_{i1} = x_{i1}, \forall i \in \{1, ..., t\}$
- $E_{is} = x_{is}, \forall i \in \{1, ..., t\}$

We are finally in conditions of formulate the constraints regarding the labour rules. Start considering the following set of constraints:

$$B_{ij} - E_{i(j+\epsilon-1)} = 0, i \in \{1, ..., t\}, j \in \{1, ..., s-\epsilon+1\}$$

These guarantee that if a team i starts a sequence of slots at j then i must finish a sequence of slots at *j+ϵ-1*, totalizing ϵ slots. Note that we cannot be sure that team i did all the ϵ slots. Something like the presented in the next table can occur:

Variables	$x_{i(j-1)}$	x_{ij}	$x_{i(j+1)}$	$x_{i(j+2)}$	$\dots$	$x_{i(j+\epsilon-2)}$	$x_{i(j+\epsilon-1)}$	$x_{i(j+\epsilon)}$
Value	0	1	1	0	0	1	1	0

In this example, $B_{ij} = E_{i(j+\epsilon-1)} = 1$, although team i did not work ϵ consecutive slots.
We can overcome this making sure that team i does not start more than one sequence of slots in each ϵ consecutive slots, which is the case of the previous example when $B_{ij} = 1$ and $B_{i(j+\epsilon-2)} = 1$.

To overcome this limitation and simultaneously satisfy the second labour rule, consider the following constraints:

$$\sum_{j=k}^{k+\epsilon+\delta-1} B_{ij} \leq 1; i \in \{1, \dots, t\}, j \in \{1, \dots, s-\epsilon-\delta+1\} \tag{2}$$

In each set of $\epsilon + \delta$ consecutive slots, (1) guarantees that a team can't start more than one sequence of slots. In particular for each set of ϵ consecutive slots, so we successfully overcame the constraint (1) limitation and the first labour rule is now completly treated.

On the other hand, if a team starts a sequence of slots at j, constraint (2) guarantees that it cannot start another one before doing all the sequence of ϵ slots and resting for δ consecutive ones, satisfying the second labour rule.

We finished this way the explanation over the additional constraints considered in this model and present the full model bellow.

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{s} r_j v_j \\
\text{Subject to:} \quad & \sum_{i=1}^{t} x_{ij} \geq 1, \quad \forall j \in \{1, ..., s\} \\
& Lb \leq \sum_{j=1}^{s} x_{ij} \leq Ub, \quad \forall i \in \{1, ..., t\} \\
& v_j \geq d_j - \sum_{i=1}^{t} x_{ij}, \quad \forall j \in \{1, \ldots, s\} \\
& x_{i(j+1)} - x_{ij} = B_{i(j+1)} - E_{ij}, \quad \forall i \in \{1, ..., t\}, \quad \forall j \in \{1, ..., s-1\} \\
& B_{i(j+1)} + E_{ij} \leq 1, \quad \forall i \in \{1, ..., t\}, \quad \forall j \in \{1, ..., s-1\} \\
& B_{i1} = x_{i1}, \quad \forall i \in \{1, ..., t\} \\
& E_{is} = x_{is}, \quad \forall i \in \{1, ..., t\} \\
& B_{ij} - E_{i(j+\epsilon-1)} = 0, \quad i \in \{1, ..., t\}, \quad j \in \{1, ..., s-\epsilon+1\} \\
& \sum_{j=k}^{k+\epsilon+\delta-1} B_{ij} \leq 1; \quad i \in \{1, \ldots, t\}, \quad j \in \{1, \ldots, s-\epsilon-\delta+1\} \\
& x_{ij} \in \{0, 1\}, \quad i \in \{1, ..., t\}, \quad j \in \{1, ..., s\} \\
& v_j \geq 0, \quad \forall j \in \{1, \ldots, s\} \\
& B_{ij} \in \{0, 1\}, \quad i \in \{1, ..., t\}, \quad j \in \{1, ..., s\} \\
& E_{ij} \in \{0, 1\}, \quad i \in \{1, ..., t\}, \quad j \in \{1, ..., s\}
\end{aligned}
$$

6 Computational Complexity

Another objective we had was to classify the problem according to its computational complexity, the intuition being that the timetable problem is NP-hard and, therefore, much more difficult to solve and much more resource-consuming than the simpler models that were discussed before, specifically models M0 and M1, which belong to the P class of problems.

The following exposition is an abridged version of chapters 6 and 10 of Schrijver's *A Course in Combinatorial Optimization*, which we highly recommend.

We start, in an informal way, with understanding what a problem is and how can we classify them. A problem is, informally, a question for which we seek answers, for example, the different problems we have modeled in the previous sections. We will only focus on decision problems, that is, *yes* or *no* problems, since most problems in Optimization, specifically the problem at hand, can be studied by the related decision problem. Therefore, we will study problems of the form: Given a certain object, does it have a certain property?

There are several ways to classify problems, but we will only consider the temporal complexity of the problem, that is, the number of steps required by algorithms (that may or may not exist) to solve the problem or check its answers. We will focus on the P, NP, NP-hard and NP-complete classes:

- A problem is in P if there exists an algorithm that solves the problem in polynomial time, that is, its running time is upper bounded by a polynomial expression in the size of the input for the algorithm. Problems in P are considered *efficiently*, *solvable* or *fast*.
- The NP class is a larger class of problems that includes all the problems in P, and most problems in Combinatorial Optimization. The letters NP stand for *non-deterministic polynomial time*. Informally, a problem is in NP if, for any input that has a positive answer, there exists a *certificate* from which the correctness of this answer can be derived in polynomial time. It is usually said that these are problems whose solutions can be quickly verified by a computer. It is also unknown if $P = NP$ or $P \neq NP$.
- NP-hard problems are at least as hard as every problem in NP: a problem Π is in NP-hard if every problem in NP can be reduced to Π, in polynomial time. NP-complete problems are those problems that are both in NP and in NP-hard. If an NP-complete problem is proved to be solvable in polynomial time, then every problem in NP can be solved in polynomial time, that is, $P = NP$.

6.1 Words, Languages and Algorithms

For a computer to solve a problem, it is necessary to properly encode the problem by describing it by a sequence of symbols from a finite alphabet Σ, for example, the ASCII set of symbols or the set $\{0, 1\}$.

Fix an alphabet Σ. We call any ordered finite sequence of elements of Σ a *word*. The set of all words is denoted by Σ^*. A language is a subset of Σ^*, that is, it is a set of words, that may have specific properties or follow a certain set of rules. The size of a word,$|.|$, is the number of symbols used in the word, counting multiplicities. Since the object of our study is solvability in polynomial time, most encodings are equivalent in this setting.

Thus, we have the following mathematical definition of problem: a problem Π is a subset of Σ^*. The informal *problem* associated with Π is: Given a word w, does w belong in Π? In this context, the word w is called an instance or the input.

We can also formally define an algorithm. Since an algorithm is a finite list of instructions to solve a problem, and these instructions are of the form *Replace subword u by v*, we can fully describe it by a finite sequence $((u_1, u_1'), \ldots, (u_n, u_n'))$, where $u_1, u_1', \ldots, u_n, u_n'$ are words. We say that word *w'* follows from w if there exists a $j \in \{1, \ldots, n\}$ such that $w = tujv$ and$w' = tu'jv$, for certain words t and v, in such a way that j is the smallest index for which this is possible and the size of t is as small as possible. The algorithm stops at word w if w has no subword in $\{u_1, ..., u_n\}$. So for any word w, either there is a unique word w' that follows from w, or the algorithm stops at w. A (finite or infinite) sequence of words $\{w_0, w_1, w_2, \ldots\}$ is said to be allowed if each w_{i+1} follows from w_i and, if the sequence is finite, the algorithm stops at the last word of the sequence. So for each word w there is a unique allowed sequence starting with w. We say that the algorithm accepts w if this sequence is finite.

Let A be an algorithm and let $\Pi \subseteq \Sigma^*$ be a problem. We say that A solves Π if Π is the set of words accepted by A. Moreover, A solves Π in polynomial-time if there exists a polynomial $p(x)$ such that, for any word $w \in \Sigma^*$, if A accepts w, then the allowed sequence starting with w contains at most $p(size(w))$ words. Thus, we can decide in polynomial time if a given word w belongs to Π. We just take $w_0 := w$, and, for $i \in \{0, 1, 2, \ldots\}$, we replace the first subword u_j, in w_i, by u_j', for some $j \in \{1, \ldots, n\}$, thus obtaining w_{i+1}. If within $p(|w|)$ iterations we stop, we know that w belongs to Π, and otherwise we know that w does not belong to Π.

6.2 The classes NP, NP-hard and NP-complete

We now give a formal definition of the class NP of problems: it consists of the problems $\Pi \subseteq \Sigma^*$ for which there exists a problem $\Pi' \in P$ and a polynomial $p(x)$ such that, for any $w \in \Sigma^*$, $w \in \Pi$ if and only if there exists a word v such that $(w, v) \in \Pi'$ and such that $|v| \leq p(|w|)$. The word v is the so-called certificate showing that w belongs to Π. With the polynomial time algorithm solving Π', the certificate proves in polynomial time that w belongs to Π.

Clearly, $P \subseteq NP$, since if Π belongs to P, then we can just take the empty string as certificate for any word w to show that it belongs to Π. That is, we can take $\Pi' := \{(w,)|w \in \Pi\}$. As $\Pi \in P$, also $\Pi' \in P$.

Let Π and Π' be two problems and let A be an algorithm. We say that A is a polynomial-time reduction of Π' to Π if A is a polynomial-time algorithm (i.e., that solves Σ^*), so that for any allowed sequence starting with w and ending with v one has: $w \in \Pi'$ if and only if $v \in \Pi$. A problem Π is called NP-complete, if $\Pi \in NP$ and for each problem Π' in NP there exists a polynomial-time reduction of Π' to Π. A problem Π is called NP-hard if it is both in NP and in NP-complete.

6.3 Matroids and Optimization Problems

In order to prove that the problems which are represented by models M0 and M1 are in P, we introduce the concept of matroid, along with some theorems, that will allow us to see that the structures for models M0 and M1 are some of those for which the greedy algorithm leads to an optimal solution, in polynomial time.

Let X be a finite set and let I be a collection of subsets of X. Then the pair (X, I) is called a *matroid* if it satisfies the following conditions:

1. $\emptyset \in I$;
2. If $Y \in I$ and $Z \subseteq Y$, then $Z \in I$;
3. If $Y, Z \in I$ and $|Y| < |Z|$, then $Y \cup \{x\}$, for some $x \in Z \setminus Y$.

For any matroid $M = (X, I)$, a subset Y of X is called *independent* if Y belongs to I, and *dependent* otherwise.

Let $Y \subseteq X$. A subset B of Y is called a *basis of* Y if B is an inclusion wise maximal independent subset of Y. That is, for any set $Z \in I$ with $B \subseteq Z \subseteq Y$ one has $Z = B$.

We next show that the matroids indeed are those structures for which the greedy algorithm leads to an optimal solution. Let X be some finite set and let I be a collection of subsets of X satisfying (1) and (2). For any weight function $w : X \to R$ we want to find a set Y in I minimizing

4. $w(Y) := \sum_{y \in Y} w(y)$

The greedy algorithm consists of selecting $y_1, ..., y_r$ successively as follows. If $y_1, ..., y_k$ have been selected, choose $y \in X$ so that:

5. (i) $y \notin \{y_1, \ldots, y_k\}$ and $\{y_1, \ldots, y_k, y\} \in I$;

   (ii) $w(y)$ is as small as possible among all y satisfying (i).

We stop if no y satisfying 5) (i) exist, that is, if $\{y_1, \ldots, y_k\}$ is a basis. Thus, we have the following statement, which proof will be omitted:

The pair (X, I) satisfying 1) and 2) is a matroid if and only if the greedy algorithm leads to a set Y in I of minimum weight $w(Y)$, for each weight function $w : X \longrightarrow \Re_+$.

We now show that the structure of model M0 is a matroid: Let $X_1, \ldots, X_m$ be subsets of the finite set X. A set $Y = \{y_1, \ldots, y_n\}$ is called a partial transversal of $X_1, \ldots, X_m$, if there exist distinct indices $i_1, \ldots, i_n$ so that $y_j \in X_{ij}$ for $j = 1, \ldots, n$. A partial transversal of cardinality m is called a *transversal*.

Another way of representing partial transversals is as follows. Let G be the bipartite graph with vertex set $V := \{1, \ldots, m\} \cup X$ and with edges all pairs $\{i, x\}$ with $i \in \{1, \ldots, m\}$ and $x \in X_i$ (we assume here that $\{1, \ldots, m\} \cap X = \emptyset$). For any matching M in G, let $\rho(M)$ denote the set of those elements in X that belong to some edge in M. Then $Y \subseteq X$ is a partial transversal if and only if $Y = \rho(M)$ for some matching M in G. Now let I be the collection of all partial transversals for $X_1, \ldots, X_m$. Then (X, I) is a matroid.

Any matroid obtained in this way, or isomorphic to such a matroid, is called a transversal matroid. If the sets $X1, \ldots, Xm$ form a partition of X, one speaks of a partition matroid. Thus, the structure of model M0 is a partition matroid, therefore the greedy algorithm will solve the problem in polynomial time.

Regarding model M1, its structure is not a matroid, but it is in fact an intersection of matroids.

Let $M_1 = (X, I_1)$ and $M_2 = (X, I_2)$ be two matroids, on the same set X. Consider the collection $I_1 \cap I_2$ of common independent sets. The pair $(X, I_1 \cap I_2)$ is generally not a matroid again. But, according to *Edmonds [1970]*, a minimum-cardinality common independent set in two matroids can be found in polynomial time.

Let $G = (V, E)$ be a bipartite graph, with colour classes V_1 and V_2, say. Let I1 be the collection of all subsets F of E so that no two edges in F have a vertex in V_1 in common. Similarly, let I_2 be the collection of all subsets F of E so that no two edges in F have a vertex in V_2 in common. So both (X, I_1) and (X, I_2) are partition matroids. Now $I_1 \cap I_2$ is the collection of matchings in G. Finding a minimum-weight common independent set amounts to finding a minimum-weight matching in G.

Thus, the structure of model M1 is an intersection of matroids, hence there exists an algorithm that solves the problem in polynomial time.

7 Computational Results

In order to prove empirically the quality of the work done, we present in this section the computational results and their analysis.

The data considered is related to TAP(Transportes Aéreos Portugueses), featuring 20 teams and 96 slots, and we used the optimizer Gurobi, with R as interface, to solve the formulated models.

There were no detailed information about the problem, so the demands, d_j and the costs per unity of demand not satisfied, c_j, were considered as random generated and fixed for all problems and the costs of assigning each team to each slot, c_{ij}, was considered unitary, for all $i \in \{1, \ldots, 20\}, j \in \{1, \ldots, 96\}$. Also the following parameters were considered:

- $Ub = 25$;
- $Lb = 10$;
- $\beta = 6$;
- $\delta = 10$

We compare at Table 1, the size of each model and, as expected, model 1 is larger in terms of constraints, since it features two additional constraints by team related to Lb and Ub, which gives 40 additional constraints.

Models	Number of variables	Constrains
0	1920	2016
1	1920	2056

We present now the outputs from R regarding the resolution of the specified models 0 and 1, respectively.

Solution to model 0

```
Warning for adding variables: zero or small (< 1e-13) coefficients, ignored
Optimize a model with 96 rows, 1920 columns and 1920 nonzeros
Coefficient statistics:
  Matrix range    [1e+00, 1e+00]
  Objective range [1e+00, 1e+00]
  Bounds range    [1e+00, 1e+00]
  RHS range       [1e+00, 1e+00]
Found heuristic solution: objective 96
Presolve removed 96 rows and 1920 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds
Thread count was 1 (of 2 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 9.600000000000e+01, best bound 9.600000000000e+01, gap 0.0%
```

Solution to model 1

```
Warning for adding variables: zero or small (< 1e-13) coefficients, ignored
Optimize a model with 136 rows, 1920 columns and 5760 nonzeros
Coefficient statistics:
  Matrix range    [1e+00, 1e+00]
  Objective range [1e+00, 1e+00]
  Bounds range    [1e+00, 1e+00]
  RHS range       [1e+00, 3e+01]
Found heuristic solution: objective 200
Presolve time: 0.03s
Presolved: 136 rows, 1920 columns, 5760 nonzeros
Variable types: 0 continuous, 1920 integer (1920 binary)

Root relaxation: cutoff, 19 iterations, 0.06 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0     cutoff    0       200.00000  199.00020  0.50%     -    0s

Explored 0 nodes (19 simplex iterations) in 0.13 seconds
Thread count was 2 (of 2 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 2.000000000000e+02, best bound 2.000000000000e+02, gap 0.0%
```

This models are very similar and, as observed in the previous chapter, solvable in polynomial time, which stands by the output obtained where Gurobi solved both problems in pre processing, since both ILP are equivalent to solve their linear relaxation. About the solution, model 1 extends model 0 to the bounds of use of each team (Lb, Ub), so naturally being more constrained, presents a solution with a higher cost.

Passing to more sophisticated models, we now present in Table 2 the models 2 and 3 sizes.

Models	Number of variables	Constrains
2	2016	2228
3	5856	15108

As we can see, labour rules induce so much complexity into our models that Model 3 features almost three times more variables and 7 times more constraints compared to Model 2.

Down below we present the output obtained solving Model 2 and 3 using the chosen software.

Solution to model 2

```
Warning for adding variables: zero or small (< 1e-13) coefficients, ignored
Optimize a model with 328 rows, 2016 columns and 7872 nonzeros
Coefficient statistics:
  Matrix range    [1e+00, 1e+00]
  Objective range [2e-02, 2e+00]
  Bounds range    [1e+00, 1e+00]
  RHS range       [2e-01, 3e+01]
```

```
Found heuristic solution: objective 185.046
Presolve removed 100 rows and 4 columns
Presolve time: 0.04s
Presolved: 228 rows, 2012 columns, 7692 nonzeros
Variable types: 0 continuous, 2012 integer (1920 binary)

Root relaxation: objective 7.581308e-01, 412 iterations, 0.02 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.75813    0   80  185.04583    0.75813   100%     -    0s
H    0     0                      14.3885680    0.75813  94.7%     -    0s
H    0     0                       5.0341756    0.75813  84.9%     -    0s
H    0     0                       3.9613256    0.75813  80.9%     -    0s
*    0     0               0       3.1711980    3.17120  0.00%     -    0s

Cutting planes:
  Gomory: 1
  MIR: 75

Explored 0 nodes (1025 simplex iterations) in 0.23 seconds
Thread count was 2 (of 2 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 3.171197997600e+00, best bound 3.171197997600e+00, gap 0.0%
```

Solution to model 3

```
Warning for adding variables: zero or small (< 1e-13) coefficients, ignored
Optimize a model with 7608 rows, 5856 columns and 48912 nonzeros
Coefficient statistics:
  Matrix range    [1e+00, 1e+00]
  Objective range [2e-02, 2e+00]
  Bounds range    [1e+00, 1e+00]
  RHS range       [2e-01, 3e+01]
Found heuristic solution: objective 138.421
Presolve removed 3780 rows and 1884 columns
Presolve time: 3.44s
Presolved: 3828 rows, 3972 columns, 41072 nonzeros
Variable types: 0 continuous, 3972 integer (3880 binary)

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   6.626875e+02   0.000000e+00      5s
   10012    9.4267501e+00   3.046524e+04   0.000000e+00     10s
   14722    2.0970197e+01   0.000000e+00   0.000000e+00     13s

Root relaxation: objective 2.097020e+01, 14722 iterations, 7.32 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
```

```
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   20.97020    0  349  138.42085   20.97020  84.9%     -   14s
H    0     0                       31.9104622   20.97020  34.3%     -   14s
H    0     0                       30.4304984   20.97020  31.1%     -   14s
H    0     0                       27.4820662   20.97020  23.7%     -   14s
     0     0   25.11592    0  363   27.48207   25.11592  8.61%     -   14s
H    0     0                       26.8460522   25.11592  6.44%     -   14s
H    0     0                       26.2885566   25.11592  4.46%     -   14s
     0     0   25.66043    0  344   26.28856   25.66043  2.39%     -   15s
     0     0   25.66043    0  344   26.28856   25.66043  2.39%     -   15s
     0     0   25.66043    0  337   26.28856   25.66043  2.39%     -   15s
     0     0   25.66043    0  323   26.28856   25.66043  2.39%     -   15s
     0     0   25.66043    0  327   26.28856   25.66043  2.39%     -   16s
     0     0   25.66043    0  385   26.28856   25.66043  2.39%     -   16s
     0     0   25.66043    0  385   26.28856   25.66043  2.39%     -   16s
     0     0   25.66043    0  125   26.28856   25.66043  2.39%     -   17s
     0     0   25.66043    0  135   26.28856   25.66043  2.39%     -   17s
     0     0   25.66043    0  152   26.28856   25.66043  2.39%     -   18s
     0     0   25.66043    0  152   26.28856   25.66043  2.39%     -   19s
     0     2   25.66043    0  152   26.28856   25.66043  2.39%     -   20s
   137   120   26.17118   46  180   26.28856   25.66043  2.39%  58.8   25s
   342   226     cutoff   61        26.28856   25.66043  2.39%  48.8   30s
   588   380   26.27545   40  147   26.28856   25.66043  2.39%  43.3   35s
   731   496   26.01215   64  152   26.28856   25.66043  2.39%  41.4   44s
   733   497   25.81668   13  305   26.28856   25.66043  2.39%  41.3   50s

Cutting planes:
  Gomory: 29
  MIR: 34
  Flow cover: 2
  Zero half: 10

Explored 737 nodes (65825 simplex iterations) in 52.87 seconds
Thread count was 2 (of 2 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 2.628855661710e+01, best bound 2.628855661710e+01, gap 0.0%
```

As mentioned before, Model 3 complexity is much higher, where the software needed for the only time to use Branch and Bound and more precisely 737 nodes of it. As expected Model 3 solution have an higher cost since it satisfies the labour rules.

8 Conclusions

The results obtained in the previous section satisfy all the constrains of the respective models and, therefore, represent valid timetables. The final model is general enough to be used in real life situations (for example in airports), being only necessary to change the weight of the objective function (cost of not meeting the demand), the demands, teams maximum and minimum quantity of assigned work and values relating to the modelled labour requirements, necessary breaks and consecutive work. The proposed objective was then sucessfully accomplished.

Two possible next steps would be:

- To consider a more complex problem, with new constrains, with the purpose of a better approach to real situations. For example, to consider the availability of the workers;
- Create a user graphical interface in which the user could make the necessary changes to a given solution and immediately see the result in the form of a timetable and its associated cost.

References

[1] http://homepages.cwi.nl/~lex/files/dict.pdf

[2] http://www.fc.up.pt/mat/3imw/P2.html

[3] http://publications.lib.chalmers.se/records/fulltext/185558/185558.pdf

Appendices

In this appendix we will show the ***R*** code that we have used to get the numerical results presented in this paper.

A R code to model 0

```
############################PROBLEM v1#####################################
#Tab problem for the day with 15 minutes time intervals
#TOTAL TEAMS = 20
#TOTAL SLOTS = 96
###OBJECTIVE FUNCTION:
#COST OF ASSIGNING EACH TEAM TO EACH SLOT IS 1 (EASY EXAMPLE)
###CONSTRAINTS:
#EACH SLOT MUST BE COVERED BY AT LEAST 1 TEAM

#Number teams
t = 20

#Number slots
s = 96

#Minimum teams per slot
cover = 1

model1 <- list()

#Costs vector
cost = as.matrix(rep(1, times = t * s))
model1$obj = cost

#Model objective
model1$modelsense = "min"

#Constrains type
model1$sense = c('>=')

#Constrains Matrix
i = 0
counter = 0
A = matrix(rep(0, times = s*s*t), nrow = s, ncol = s*t)
for(j in 1:s){
  for(i in 1:t){
    A[j, 1 + (i-1)*s+counter] = 1;
  }
  counter = counter + 1;
}
model1$A = A

#Constrains independent vector
model1$rhs = as.matrix(rep(cover, times = s))
```

```
#Variables type
model1$vtypes = as.matrix(rep('B', times = s*t))

#Gurobi call
result1 <- gurobi(model1)
print(result1$x)
```

B R code to Model 1

```
############################PROBLEM v2#####################################
#Tab problem for the day with 15 minutes time intervals
#TOTAL TEAMS = 20
#TOTAL SLOTS = 96
###OBJECTIVE FUNCTION:
#COST OF ASSIGNING EACH TEAM TO EACH SLOT IS 1 (EASY EXAMPLE)
###CONSTRAINTS:
#EACH SLOT MUST BE COVERED BY AT LEAST 1 TEAM
#EACH TEAM MUST BE USED AT LEAST 10 TIMES
#EACH TEAM MUST BE USED AT MOST 25 TIMES

#Number teams
t = 20

#Number slots
s = 96

#Minimum usage of each team
LB<-10

#Maximum usage of each team
UB<-25

#Minimum teams per slot
Cover<-1

model1 <- list()

#Cost vector
cost = as.matrix(rep(1, times = t * s))
model1$obj = cost

#Problem objetive
model1$modelsense = "min"

#Constrains type
model1$sense = c('>=')

#Constrains matrix
i = 0
counter = 0
A = matrix(rep(0, times = (s+2*t)*s*t), nrow = s+2*t, ncol = s*t)
for(j in 1:s){
```

```
  for(i in 1:t){
    A[j, 1 + (i-1)*s+counter] = 1;
  }
  counter = counter + 1;
}

for(i in (s+1):(s+t)){
  A[i, (1+s*(i-(s+1))):(s+s*(i-(s+1)))] = 1
}

for(i in (s+t+1):(s+2*t)){
  A[i, (1+s*(i-(s+t+1))):(s+s*(i-(s+t+1)))] = -1
}

model1$A = A

#Constrains independent vector
model1$rhs = as.matrix(c(rep(cover, times = s),rep(LB,times=t), rep(-UB, times = t)))

#Variables type
model1$vtypes = as.matrix(rep('B', times = s*t))

#Gurobi call
result1 <- gurobi(model1)

print(result1$x)
```

C R code to Model 2

```
#####################PROBLEM V3#############################
#Tab problem for the day with 15 minutes time intervals
#TOTAL TEAMS = 20
#TOTAL SLOTS = 96
###OBJECTIVE FUNCTION:
#KNOWN COST PER MISSING SERVICE (VERSUS GAP) PER TIME SLOT
###CONSTRAINTS:
#EACH SLOT MUST BE COVERED BY AT LEAST 1 TEAM
#EACH TEAM MUST BE USED AT LEAST 10 TIMES
#EACH TEAM MUST BE USED AT MOST 25 TIMES
#EACH SLOT GAP MUST BE EQUAL TO THE DIFERENCE BETWEEN THE DEMAND AND THE SERVICE
#EACH SLOT GAP MUST BE EQUAL OR GREATER THAN 0


#Number of teams
t<-20

#Number of slots
s<-96

#Minimum usage of each team
LB<-10
```

```
#Maximum usage of each team
UB<-25

#Minimum teams per slot
Cover<-1

#Demand of each tap slot
slot_demands <- as.matrix(read.csv2("~/Mestrado/2Ano/IMW/Tap_Demans.txt", header=FALSE,_
sep=""))

#Each slot gap cost
gap_costs <- as.matrix(read.csv2("~/Mestrado/2Ano/IMW/TAP_Costs.txt", header=FALSE,_
sep=""))


model1 <- list()

#Cost vector
cost = as.matrix(c(rep(0, times = t * s), gap_costs))
model1$obj = cost

#Problem type
model1$modelsense = "min"

#Constraints type
model1$sense = c('>=')

#Constrains Matrix construction
i = 0
counter = 0
SLOT_COVERAGE = matrix(rep(0, times = s*s*t), nrow = s, ncol = s*t)
for(j in 1:s){
  for(i in 1:t){
    SLOT_COVERAGE[j, 1 + (i-1)*s+counter] = 1;
  }
  counter = counter + 1;
}

TEAM_USAGE_LOWER_BOUND = matrix(rep(0, times=t*s*t), nrow = t, ncol = s*t)
for(i in 1:t){
  TEAM_USAGE_LOWER_BOUND[i, (1 + (i-1)*s):(s + (i-1)*s)] = 1
}

TEAM_USAGE_UPPER_BOUND = matrix(rep(0, times = t * s * t), nrow = t, ncol = s*t)
for(i in 1:t){
  TEAM_USAGE_UPPER_BOUND[i, (1 + (i-1)*s):(s + (i-1)*s)] = -1
}

TEAM_USAGE = rbind(TEAM_USAGE_LOWER_BOUND, TEAM_USAGE_UPPER_BOUND)

IDENTITY = diag(x = 1, nrow = s)
```

```
C = matrix(rep(0, times = s*s*t), nrow = s, ncol = s*t)

A = rbind(SLOT_COVERAGE, TEAM_USAGE)
B = matrix(rep(0, times = s*(s+2*t)), nrow = s+2*t, ncol = s)

CONSTRAINT_MATRIX = rbind(cbind(A, B),cbind(SLOT_COVERAGE, IDENTITY), cbind(C, IDENTITY))
model1$A = CONSTRAINT_MATRIX

#Constrains independent vector
model1$rhs = as.matrix(c(rep(Cover, times = s),rep(LB,times=t), rep(-UB, times = t),_
slot_demands, rep(0, times = s)))

#Variably types
model1$vtypes = as.matrix(c(rep('B', times = s*t), rep('C',times = s)))

#Gurobi call
result1 <- gurobi(model1)

print(result1$x)
```

D R code to Model 3

```
#####################PROBLEM V3#############################
#Tab problem for the day with 15 minutes time intervals
#TOTAL TEAMS = 20
#TOTAL SLOTS = 96
###OBJECTIVE FUNCTION:
#KNOWN COST PER MISSING SERVICE (VERSUS GAP) PER TIME SLOT
###CONSTRAINTS:
#EACH SLOT MUST BE COVERED BY AT LEAST 1 TEAM
#EACH TEAM MUST BE USED AT LEAST 10 TIMES
#EACH TEAM MUST BE USED AT MOST 25 TIMES
#EACH SLOT GAP MUST BE EQUAL TO THE DIFERENCE BETWEEN THE DEMAND AND THE SERVICE
#EACH SLOT GALP MUST BE EQUAL OR GREATER THAN 0
#EACH TEAM MUST HAVE A BREAK OF DELTA SLOTS AFTER BEING ASSIGNED FOR A JOB (SLOT OR
# SET OF CONSECUTIVE SLOTS)
#WHEN A TEAM IS ASSIGNED TO  SLOT, IT WORKS BETA SLOTS IN A ROW

#Number of teams
t<-20

#Number of slots
s<-96

#Minimum usage of each team
LB<-10

#Maximum usage of each team
UP<-25

#Minimum teams per slot
Cover<-1
```

```
#Size of necessary break
min_team_break<-10

#Consecute ssigned slots constraint
consecutive_slots<-6

#Demand of each tap slot
slot_demands <- as.matrix(read.csv2("~/Mestrado/2Ano/IMW/Tap_Demans.txt", header=FALSE,_
sep=""))

#Each slot gap cost
gap_costs <- as.matrix(read.csv2("~/Mestrado/2Ano/IMW/TAP_Costs.txt", header=FALSE,_
sep=""))

model1 <- list()

#Cost vector
cost = as.matrix(c(rep(0, times = t * s), gap_costs, rep(0, times = 2*s*t)))
model1$obj = cost

#Problem type
model1$modelsense = "min"


#Constrain Matrix construction
i = 0
counter = 0
SLOT_COVERAGE = matrix(rep(0, times = s*s*t), nrow = s, ncol = s*t)
for(j in 1:s){
  for(i in 1:t){
    SLOT_COVERAGE[j, 1 + (i-1)*s+counter] = 1;
  }
  counter = counter + 1;
}

TEAM_USAGE_LOWER_BOUND = matrix(rep(0, times=t*s*t), nrow = t, ncol = s*t)
for(i in 1:t){
  TEAM_USAGE_LOWER_BOUND[i, (1 + (i-1)*s):(s + (i-1)*s)] = 1
}

TEAM_USAGE_UPPER_BOUND = matrix(rep(0, times = t * s * t), nrow = t, ncol = s*t)
for(i in 1:t){
  TEAM_USAGE_UPPER_BOUND[i, (1 + (i-1)*s):(s + (i-1)*s)] = -1
}

TEAM_USAGE = rbind(TEAM_USAGE_LOWER_BOUND, TEAM_USAGE_UPPER_BOUND)

B = matrix(rep(0, times = s*(s+2*t)), nrow = s+2*t, ncol = s)

C = matrix(rep(0, times = s*s*t), nrow = s, ncol = s*t)
```

```
IDENTITY = diag(x = 1, nrow = s)

A = rbind(SLOT_COVERAGE, TEAM_USAGE)

CONSTRAINT_MATRIX_v1 = rbind(cbind(A, B),cbind(SLOT_COVERAGE, IDENTITY),_
cbind(C, IDENTITY))

number_previous_constraints = nrow(CONSTRAINT_MATRIX_v1);

ZEROS_HELPER = matrix(rep(0, times = number_previous_constraints*2*s*t),_
nrow = number_previous_constraints, ncol = 2*s*t)

CONSTRAINT_MATRIX_v2 = cbind(CONSTRAINT_MATRIX_v1, ZEROS_HELPER)

FIRST_SLOT_BEGIN_CONSTRAINTS = matrix(rep(0, times = t*(s*t +s+2*s*t)),_
nrow = t, ncol = s*t+s+2*s*t)

for(i in 1:t){
  FIRST_SLOT_BEGIN_CONSTRAINTS[i, 1 + (i-1)*s] = 1
  FIRST_SLOT_BEGIN_CONSTRAINTS[i, s*t + s +(i-1)*s+1] = -1
}

LAST_SLOT_END_CONSTRAINTS = matrix(rep(0, times = t*(s*t +s+2*s*t)),_
nrow = t, ncol = s*t+s+2*s*t)

for(i in 1:t){
  LAST_SLOT_END_CONSTRAINTS[i, s+(i-1)*s] = 1
  LAST_SLOT_END_CONSTRAINTS[i, s*t+s+s*t+ s+(i-1)*s] = -1
}

BEING_END_CONSTRAINTS = matrix(rep(0, times = t*(s-1)*(s*t+s+2*s*t)),_
nrow = t*(s-1), ncol = s*t+s+2*s*t)

for(i in 1:t){
  for(k in 1:(s-1)){
    BEING_END_CONSTRAINTS[k + (s-1)*(i-1), k + (i-1)*s] = -1
    BEING_END_CONSTRAINTS[k + (s-1)*(i-1), k + 1 + (i-1)*s] = 1
    BEING_END_CONSTRAINTS[k + (s-1)*(i-1), s*t+s+k+1+(i-1)*s] = - 1
    BEING_END_CONSTRAINTS[k + (s-1)*(i-1), s*t+s+s*t+k+(i-1)*s] = 1
  }
}

ALL_BEGIN_END_CONSTRAINTS = rbind(FIRST_SLOT_BEGIN_CONSTRAINTS,_
LAST_SLOT_END_CONSTRAINTS, BEING_END_CONSTRAINTS)

number_of_begin_end_constraints = nrow(ALL_BEGIN_END_CONSTRAINTS);

CONSTRAINT_MATRIX = rbind(CONSTRAINT_MATRIX_v2, ALL_BEGIN_END_CONSTRAINTS)

BEGIN_END_VARIABLES = matrix(rep(0, times = t*(s-1)*(s*t + s + 2*s*t)),_
```

```
nrow = t*(s-1), ncol= s*t + s + 2*s*t)

for(i in 1:t){
  for(k in 1:(s-1)){
    BEGIN_END_VARIABLES[k + (s-1)*(i-1), s*t+s+k+1+(i-1)*s] = 1
    BEGIN_END_VARIABLES[k + (s-1)*(i-1), s*t+s+s*t+k+(i-1)*s] = 1
    }
}

number_begin_end_variables = nrow(BEGIN_END_VARIABLES);

CONSTRAINT_MATRIX = rbind(CONSTRAINT_MATRIX, BEGIN_END_VARIABLES)

CONSECUTIVE_SLOTS_CONSTRAINTS = matrix(rep(0,_
times = t*(s-consecutive_slots+1)*(s*t + s + 2 *s*t)), nrow = t*(s-consecutive_slots+1),_
ncol = s*t + s + 2 *s*t)

for(i in 1:t){
  for(k in 1:(s-consecutive_slots+1)){
    CONSECUTIVE_SLOTS_CONSTRAINTS[k + (i-1)*(s-consecutive_slots+1),_
s*t + s + k + s*(i-1)] = 1
    CONSECUTIVE_SLOTS_CONSTRAINTS[k + (i-1)*(s-consecutive_slots+1),_
s*t + s + s*t + k - 1 + consecutive_slots + s*(i-1)] = - 1
  }
}

number_consecutive_constraints = nrow(CONSECUTIVE_SLOTS_CONSTRAINTS);

CONSTRAINT_MATRIX = rbind(CONSTRAINT_MATRIX, CONSECUTIVE_SLOTS_CONSTRAINTS)

TEAM_BREAK_CONSTRAINTS = matrix(rep(0,_
times = (t*(s-consecutive_slots - min_team_break + 1))*(s*t + s + 2*s*t)),_
nrow = t*(s-consecutive_slots - min_team_break + 1) , ncol = s*t + s + 2*s*t)

for(i in 1:t){
  for(k in 1:(s-consecutive_slots-min_team_break +1)){
    for(j in k:(k+consecutive_slots+min_team_break-1)){
      TEAM_BREAK_CONSTRAINTS[k +_
((s-consecutive_slots-min_team_break +1))*(i-1), s*t + s + j + s*(i-1)] = 1
    }
  }
}

number_team_break_constrains = nrow(TEAM_BREAK_CONSTRAINTS);

CONSTRAINT_MATRIX = rbind(CONSTRAINT_MATRIX, TEAM_BREAK_CONSTRAINTS)

model1$A = CONSTRAINT_MATRIX

#Constrains type
model1$sense = as.matrix(c(rep('>=', times = number_previous_constraints),_
```

```
rep('=', times = number_of_begin_end_constraints),_
rep('<=', times = number_begin_end_variables),_
rep('=', times = number_consecutive_constraints),_
rep('<=', times = number_team_break_constrains)))

#Constrains independent vector
model1$rhs = as.matrix(c(rep(Cover, times = s),rep(LB,times=t),_
rep(-UP, times = t), slot_demands, rep(0, times = s),_
rep(0, times = number_of_begin_end_constraints),_
rep(1, times = number_begin_end_variables),_
rep(0, times =number_consecutive_constraints),_
rep(1, times = number_team_break_constrains)))

#Variably types
model1$vtypes = as.matrix(c(rep('B', times = s*t),_
rep('C',times = s), rep('B', times = 2*s*t)))

#Gurobi call
result1 <- gurobi(model1)

X<-result1$x
View(as.matrix(X))
```