Universitá degli Studi di Napoli Federico II

Dottorato di Ricerca

in

Scienze Matematiche e Informatiche

XXXI ciclo

# A Combinatorial Optimization Approach to Accessibility Services in International Airports

*Michele Mele*

Anno Accademico 2017/2018

*" The only disability is when people cannot see human potential "*

Debra Ruh, founder of TecAccess advocate for the rights of people with special needs.

# Contents

# Abstract

In this PhD thesis we study a specific variant of the well known Fixed Job Scheduling Problem, namely the Tactical Fixed Job Scheduling Problem with Spread-Time constraints. In this problem it is required to schedule a number of jobs on non identical machines that differ from each other for the set of jobs they can perform and that have constraints on the length of their duty. After providing an extensive literature review of the Fixed Job Scheduling and of its main variants, the original contribution is presented. We illustrate some lower bounds for the optimal value of the problem and display the first heuristic algorithm for solving it. We also study a specific case of interest connected with the assistance of passengers with special needs in large scale international airports.

# Chapter 1

# Introduction

## 1.1 Motivation for this research

Today air transport has become the quickest and safest form of long distance transport. In the last three years the record of the busiest day in aviation history has been broken 11 times and 83 of the 100 busiest days in civil aviation history have been recorded in the last 6 years. In fact the last time in which that record has been broken was on 13 July 2018 with 205468 flights registered.

However passengers with special needs (for example blind people, partially sighted people, people in wheelchair) still suffer remarkable inconveniences when they have to take a flight, especially when they have to start a journey by their own or when they have to change flight at a transition airport. Many airports and airline companies are planning to organize assistance services for these passengers to allow them to travel in autonomy.

In the last decades many rules and conventions have been stated to regulate such services. Unfortunately the road to accessibility is still long to go: in almost every international airport the assistance services use to violate some of the rules

stated in the international treaties. So one may ask for the minimum number of workers needed to provide assistance and accompany passengers with special needs respecting all the stated rules. The main rules stated by the international treaties are:

- all passengers with special needs must be assisted by the same worker during their whole permanence in the airport;
- the worker assisting a passenger has to speak a language comprehensible by him/her;
- the basic work rules must be respected, especially the rule on the length of a day duty.

This real world problem can be seen as a Tactical Fixed Job Scheduling with Spread-Time constraints (TFJSS), a variant of the well known Fixed Job Scheduling (or Fixed Interval Scheduling). In this problem a number of jobs must be processed on non identical machines with a limit on the availability of the machines (spread-time). The machines are not equal because they can process different jobs and therefore they are divided into classes. In this scenario the workers of the airport are identified with the machines, the lenght of the day duty coincides with the spread-time and workers are not equal because they can speak different languages and so assist different passenger groups.

## 1.2 Outline

In Chapter 2 it is first introduced the basic Fixed Job Scheduling with the exact algorithms to solve it (section 2.1). Subsequently we analize the literature of the main variants of FJS, starting with the Fixed Job Scheduling with Spread-Time

constraints (section 2.2). Then we analize the variants in which machines differ from each other for the sets of jobs they can execute; the results for the Tactical Fixed Job Scheduling are presented (section 2.3) before reviewing the only exact algorithm for the Fixed Job Scheduling with Spread-Time constraints (section 2.4).

In Chapter 3 the original contribution is presented. After proving the intractability of the problem (section 3.2) we present two lower bounds for the optimal value of TFJSS (section 3.3). Then we present the first heuristic algorithm for TFJSS and analize in detail the results of the computational experiments comparing them with the real world cases detected in large scale international airports (section 3.4).

# Chapter 2

# Literature review

## 2.1 The basic fixed job scheduling

In an instance of the basic Fixed Job Scheduling Problem (FJS) it is asked to find the minimum number of machines needed to process $n$ jobs $J_j$ with fixed release time $r_j$ and deadline $d_j$, $(j = 1, ..., n)$. Each machine is able to process only a job at a time and it has to complete the processing of a job after starting it with no interruption. All the machines are available along the whole (discrete) interval $[0, T]$, with $T = \max d_j$. Two jobs $J_j$ e $J_k$ are said to be incompatible if $r_j \leq r_k \leq d_j$.

This is a special case of the Dilworth's problem:

let $N$ is a finite partially ordered set with elements $1, ..., n$ with a strict order relation "$<$". A sequence of elements of $N$ $i_1, ..., i_k$ is a chain (or a sting) if and only if $i_1 < ... < i_k$. Which is the minimum number of chains needed to cover $N$?

The FJS can be seen as a Dilworth's problem in which we explicit the constraints due to the incompatibility of the jobs. If $N$ is the set of all jobs $J_j$ $(j = 1, ..., n)$

an order relation can be defined putting $J_j < J_k$ if and only if $d_j \leq r_k$. The jobs belonging to the same chain can be assigned to an unique machine. So one can ask to find the minimum number of chains (machines) needed to cover $N$. The jobs in $S = \{r_j, d_j : J_j \in N\}$ can be numbered so that $J_k \leq J_j$ implies that if $J_l < J_k$ then $J_l < J_j$. We obtain such a relation ordering jobs in non decreasing $r_j$ , i.e. $J_k \leq J_j$ if and only if $r_k \leq r_j$. The authors in [1] introduce an algorithm that constructs an optimal solution of FJS. This procedure, called *staircase rule*, starts from the job with lower index (after the above enumeration) and repeatedly selects the successor of lower index of the last selected element until there are no such successors anymore. The constructed chain is deleted and the process restarts.

Dilworth's theorem states that for each partially ordered set the minimum number of chains needed to cover the set equals the maximum number of pairwise non comparable elements. The authors in [2] introduce an alternative algorithm to comput the optimal value of FJS. They introduce a function (step function) defined as follows: $f_S(0) = 0$, the value of the function increases by 1 when a job starts and it decreases by 1 when it ends. The value of $f_S(t)$ is exactly the number of jobs active at time $t$. A more formal definition of step function can be given introducing the function $x_j(t)$ assuming value 1 when $r_j \leq t < d_j$ and value 0 otherwise. So the step function can be defined as follows:

$$f_S(t) = \sum_{J_j \in N} x_j(t).$$

This function is defined on $[0, T]$ where $T$ is the maximum of the $d_j$s. Let $M = \max f_S(t)$. The region $[t_1, t_2)$ will be called a *maximum interval* if $f_S(t) = M$ with $t \in [t_1, t_2)$ and no job starts or ends in this interval. Now enumerate the maximum intervals from left to right and denote the $i-$th maximum interval by $R_i = [p_i, q_i)$. Note that the extremes of a maximum interval correspond to the release time or

the deadline of some job (not necessarily the same one), and that, if at a given time $t$ a job starts and another ends, the regions that immediately precede and follow $t$ are numbered in a different way. We also define the *hollow* regions $H_0 = [0, p_1]$, $H_1 = [q_1, p_2]$,...,$H_{k-1} = [q_{k-1}, p_k]$, $H_k = [q_k, T]$. A job $J_j$ is said to be "of the fist type" if there exists a hollow region in which it starts and ends; a job that is not of the first type is said to be "of the second type". If $\alpha$ is a subset of jobs, $S - \alpha$ will be the instance obtained removing the jobs of $\alpha$ from the initial instance.

**Theorem 1** ([2]). *Given an instance $S$ of FJS, the minimum number of machines needed to process all jobs is $z(S) = \max\{f_S(t) | t \in [0, T]\}$. Moreover there exist $z(S)$ disjoint strings such that each job is in exactly one string.*

*Proof.* If a job $J_j$ of the first type is removed, we get a new step function $f_{S-J_j}(t)$ e $\{\max f_{S-J_j}(t) : t \in [0, T]\} = \max\{f_S(t) | t \in [0, T]\}$. Removing a job with release time and deadline in two different hollow region brings to the reduction of the value of the step function in every region in which the job is active. We now build a string $\alpha_1$ as follows: we start selecting a job $J_{j_1}$ with $r_{j_1} \in H_0$; if this is a job of the first type, we select another $J_{j_2}$ with $r_{j_2} \in H_0$ and $r_{j_2} \geq d_{j_1}$ and add it to the string. That job exists because the right extreme of $H_0$ is the release time of some job. Then we iterate the process until we find a job of the first type $J_{j_p}$ with $d_{j_p} \in H_m$ $(m > 0)$ and add it to the string. If $m < k$ it must exist a job $J_{j_{p+1}}$ with $r_{j_{p+1}} \geq d_{j_p}$ so that $r_{j_{p+1}} \in H_m$. The process stop when it finds a job $J_{j_q}$ with $r_{j_q} \in H_k$. Finally we remove the string $\alpha_1 = \{J_{j_1}, ..., J_{j_p}, J_{j_{p+1}}, ..., J_{j_q}\}$ from $S$ to get the new instance $S - \alpha_1$ and the new step function $f_{S-\alpha_1}(t)$. So $\{\max f_{S-\alpha_1}(t) : t \in [0, T]\} = \max\{f_S(t) - 1 : t \in [0, T]\}$. After removing $\alpha_1$ the process restarts building and then deleting a new string $\alpha_2$ until no job remains. Every time a job is removed the value of the step function decreases by 1. So we

obtain $M$ strings and the jobs in each string can be assigned to one machine. Let $z(S)$ be the minimum number of machines needed to process all jobs. By definition of the step function and of $M$, we have that $z(S) \geq M$. □

In [3] the authors introduce a procedure to construct strings in parallel instead of building them in series. They define a pile $P$ of non used machines and order the values of $r_j$ and $d_j$ in non decreasing order; so they get $2n$ values that will be indicated by $u_1, u_2, ..., u_{2n}$. For $k = 1, ..., 2n$ the algorithm in [3] takes $u_k$: if it corresponds to the release time of a job $J_j$, that job is assigned to the machine which is on the top of the pile $P$, that machine is then removed from $P$; if it corresponds to the deadline of a job $J_j$, the machine to which that job is assigned is put on the top on the pile $P$.

## 2.2 The Fixed Job Scheduling with Spread-Time Constraints

The authors in [4], [5] and [6] introduce some variants of the Fixed Job Scheduling Problem adding new real world inspired constraints. In [4] the Fixed Job Scheduling with spread-time constraints (FJSS)is introduced: an instance of FJSS consists of $n$ jobs $J_j$ $(1 \leq j \leq n)$ that must be processed without preemption from a fixed release time $r_j$ to a fixed deadline $d_j$. The jobs must be processed by identical machines that can execute only one job at a time. Each machine can only work for at most $L$ time units that must be consecutive: the spread-time $L$ is defined as the range between the start of the first job and the end of the last job assigned to a machine and it is equal for all machines. The authors in [4] and [6] introduce the decision variables $y_i$, that assume value 1 if machine $P_i$ is used and value 0

otherwise, and $x_{ij}$ assuming value 1 if $P_i$ processes job $J_j$ and value 0 otherwise. If $m$ is an upper bound for the number of machines ($m \leq n$) the problem can be modeled as follows:

$$\min z = \sum_{i=1}^{m} y_i \tag{2.1}$$

s. t.

$$x_{ij} \leq y_i \quad i = 1, ..., m; j = 1, ..., n, \tag{2.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad j = 1, ..., n, \tag{2.3}$$

$$x_{ij} + x_{ik} \leq 1 \quad i = 1, ..., m; j = 1, ..., n-1; k \in \{l > j : r_l < d_j\}, \tag{2.4}$$

$$x_{ij} + x_{ik} \leq 1 \quad i = 1, ..., m; j = 1, ..., n-1; k \in \{l : d_l > r_j + L\}, \tag{2.5}$$

$$x_{ij}, y_i \in \{0, 1\} \quad i = 1, ..., m; j = 1, ..., n, \tag{2.6}$$

The objective function (2.1) aims to minimize the number of machines needed to process all jobs. Constraints (2.2) assure that $y_i$ assumes value 1 only if $P_i$ is used. Constraints (2.3) assure that every job is assigned to exactly one machine. Constraints (2.4) assures that no machine processes two incompatible jobs. Constraints (2.5) grant that the spread-time constraint is not violated.

The problem described above is NP-hard. Consider the problem known as the Circular Arc Coloring: given $n$ arcs on a circle with the $j-$th arc having extremes $a_j$ and $b_j$ belonging to $[0, 2\pi)$ and an integer $q$, is it possible to color these arcs using at most $q$ colours so that no couple of arcs of the same colour intersects? W.l.g. assume that the $2n$ extremes of the arcs are pairwise disjoint. Given an instance of the Circular Arc Coloring we can build an instance of FJSS putting $L = 2\pi$, $r_j = a_j$, $d_j = b_j$ if $a_j < b_j$, $d_j = b_j + 2\pi$ otherwise ($j = 1, ..., n$). Thus two arcs will be intersecting if and only if the corresponding jobs either overlap or

violate the spread-time constraint. So we produce a polynomial reduction from the Circular Arc Coloring to the FJSS that is cleary NP-hard.

In the preemptive relaxation of the problem it is possible to divide a job during its execution and to assign the different parts of a job to different machines. Call $z(I)$ and $\overline{z}(I)$ the optimal values for an instance $I$ and for its preemptive relaxation respectively. Clearly we have $\overline{z}(I) \leq z(I)$.

The authors in [4] introduce two procedures, one based on a greedy criterion and one on the preemptive relaxation on the problem: those procedures will be embedded in the approximation algorithms in [4] and [6]. We will denote by PFJSS the preemptive relaxation of FJSS.

**PROCEDURE** GREEDY$(\overline{J}, a)$

**begin**

Let $\overline{J}$ be the set of unassigned jobs.

**for** $J_j \in \overline{J}$ in non decreasing order of $r_j$, **do**

**begin**

let $F$ be the set of all machines $P_i, i \leq a$ to which at least a job has already been assigned and that can process $J_j$ without violating the spread-time constraint;

**if** $F = \emptyset$, then $a = a + 1$ and assign $J_j$ to $P_a$,

**else** assign $J_j$ to one of the machines of $F$ via a criterion $C$

**end**

**end**

**PROCEDURE** PREEMPT

**begin**

$J := \{J_1, ..., J_n\}$;

$\overline{z} := 0$;

$k := 0$;

**while** $J \neq \emptyset$ do

**begin**

let $J_j$ be the job of $J$ with minimum $r_j$;

$J := J \setminus \{J_j\}$;

let $F$ be the set of all machines $P_i, i \leq \overline{z}$ that can execute the first work unit $(r_j, r_j + 1]$ of $J_j$;

**if** $F \neq \emptyset$,

**begin**

select a machine $P_i \in F$;

**if** $d_j \leq r_{f(i)} + L$ (where $f(i)$ is the index of the first job assigned to $P_i$) then assign $J_j$ to $P_i$;

**else**

**begin**

assign the first part of $(r_j, r_{f(i)} + L]$ of $J_j$ to $P_i$;

$k := k + 1$;

let $J_{n+k}$ be a new job with $r_{n+k} = r_{f(i)} + L$ and $d_{n+k} = d_j$;

$J := J \cup \{J_{n+k}\}$

**end**

**end**

**else**

**begin**

$\overline{z} := \overline{z} + 1$;

assign $J_j$ to $P_{\overline{z}}$;

$f(\overline{z}) := j$

**end**

**end**

**end**

**Theorem 2** ([6]). *Procedure PREEMPT solves PFJSS to optimality; moreover it introduces at most* $\overline{z} - 1 < n$ *preemptions.*

*Proof.* Given an instance $I$ of PFJSS, for each time $t$, let $I(t)$ be the subinstance obteined removing all jobs starting after $t$ and putting $d_j = \min(d_j, t)$ for the remaining jobs. Let $Q(t)$ be the set of all jobs active at time $t$. Since no machine working at time $t$ can process jobs (or part of a job) in $I(t-L)$, a lower bound for the optimal value of $I(t)$ can be defined by $b(t) = 0$ if $t \leq r_1$, $b(t) = |Q(t)| + b(t-L)$ otherwise. Let $u(t)$ be the number of machines processing at least a work unit in a solution of an instance $I$ given in output by PREEMPT. Thus $b(t) \leq u(t)$. The proof will be complete showing that $b(t) = u(t)$ for each $t$.

Assume by contraddiction that the last statement is not true and let $t'$ be the minimum $t$ so that $b(t) < u(t)$. Consider the iteration in which$P_{u(t')}$ is added and let $J_j$, with $r_j = t' - 1$, be the job assigned to it. So it means that the machines $P_i$ $(i \leq u(t') - 1)$ process, at time $t'$, either $Q(t')$ or part of a job in $I(t' - L)$. Hence $u(t') - 1 \leq (|Q(t')| - 1) + u(t' - L)$. By definition of $t'$, $u(t' - L) = b(t' - L)$, so $u(t') \leq b(t')$.

Note that no more than one preemption is produced for each machine. Assigning $(r_j, r_{f(i)} + L]$ to $P_i$ we avoid that every successive $J_h$ is assigned to $Pi$ since $r_h \geq r_j$. Notably no preemption is introduced for the machine processing the last job. □

The complexity of procedure PREEMPT is $O(n \log n)$. An immediate heuristic algorithm for FJSS is the following one, called HPRS: once executed PREEMPT to an instance, we introduce a new machine for each preempted job.

**Theorem 3** ([6]). *r(HPRS)=2.*

*Proof.* From the previous proof we have $r(\text{HPRS}) \leq 2$. Consider the family of instances with $n = L$ even, $r_j = j$ for each $j$, $d_j = r_j + 1$ when $j$ is odd, $d_j = r_j + L$ when $j$ is odd. The optimal solution uses a machine for each $J_j$ with $j$ even plus a machine for the remaining jobs; hence $z = \frac{n}{2} + 1$. Procedure PREEMPT introduce $\frac{n}{2} + 1$ machines preempting all jobs of even index at time $d_j - 1$. HPRS adds $\frac{n}{2}$ more machines producing a solution with value $a = n + 1$. The ratio $\frac{a}{z}$ goes to 2 as $n$ increases. □

It is possible to improve the procedure HPRS assigning the preempted jobs at the end of its execution in a new way instead of opening a new machine.

ALGORITMH SPREEMPT

begin

let $\overline{z}$ be the value of the solution found by PREEMPT and let $\overline{J}$ be the set of preempted jobs;

remove from the machines the parts of the preempted jobs;

GREEDY$(\overline{J}, \overline{z})$

end

**Theorem 4** ([6]). *r(SPREEMPT)=2.*

*Proof.* Since SPREEMPT dominates HPRS, than $r(\text{SPREEMPT}) \leq 2$. The family of instances of the previous proof shows that the worst case ratio goes to 2 as $n$ grows. □

Since GREEDY and PREEMPT can be implemented as $O(n \log n)$, SPREEMPT is $O(n \log n)$ too.

Putting $\overline{J} = J$ and $a = 0$ in GREEDY we get a new algorithm with the same worst case ratio of SPREEMPT, has proved in [6].

**Theorem 5** ([6]). *Let $J = \{J_1, ..., J_n\}$ be the set of iobs of an instance $I$ of FJSS. For each criterion $C$ used in GREEDY, let $H = \{H_1, ..., H_a\}$ be the partition of $J$ given in output, where $P_i$ processes jobs of $H_i$ $(i = 1, ..., a)$. For each $Q \subseteq \{1, ..., a\}$ let $I(Q)$ be the subinstance of $I$ defined by the set of jobs $J(Q) = J \setminus \cup_{i \in Q} H_i$. Then there exists a criterion $C_Q$ for GREEDY producing for $I(Q)$ the partition $H \setminus \{H_i : i \in Q\}$ di $J(Q)$.*

*Proof.* Let $P_{x_j}$ be the machine selected for processing $J_j$ with criterion $C$. The criterion $C_Q$ consists in selecting the machine $P_{\varrho_j}$ for each $J_j \in J(Q)$, where $\varrho_j = x_j - |\{i \in Q : i < x_j\}|$. Since $C_Q$ assigns the jobs $J_j \notin J(Q)$ to machines $P_i$ with $i \in Q$ and viceversa those machines only process jobs $J_j \notin J_Q$, this selection will always be possible and it will be leading to the required partition. □

**Theorem 6** ([6]). *r(GREEDY)=2, for every choise of $C$.*

*Proof.* First $a(I) < 2\overline{z}(I)$, where $a(I)$ is the value of the solution given by GREEDY and $\overline{z}_I$ is the optimal value of the preemptive relaxation. The proof of this statement is by induction on $\overline{z}_I$. This is true for $\overline{z}_I = 1$ since $\max\{d_j\} - r_1 \leq L$ and $r_j \geq d_{j-1}$; so $a(I) = 1$. Assume now $a(I) < 2\overline{z}(I')$ for $\overline{z}(I') \leq k$ and consider an instance $I$ with set of jobs$J$ such that $\overline{z}_I = k + 1$. Let $C$ be the selection criterion used in GREEDY and suppose that the solution with value $\overline{z}_I$ has been

obtained by PREEMPT adopting the following scheme: if some of the machines able to process the first work unit of $J_j$ can also process $J_j$ entirely, then select $P_i$ among those by criterion $C$; choose randomly otherwise. Notably GREEDY and PREEMPT start to build the same solution until a job $J_{j*}$ that is fractioned by PREEMPT is found. Let $P_\alpha$ be the machine processing the first part $J'_{j*}$ of $J_{j*}$ in the solution given by PREEMPT and $P_\beta$ be the machine processing entirely $J_{j*}$ in the solution given by GREEDY. Thus in the solution given by PREEMPT $P_\alpha$ processes jobs in $H'_\alpha = \{J_{\alpha_1}, ..., J_{\alpha_q}, J'_{j*}\}$ while in the solution given by GREEDY $P_\alpha$ processes jobs in $H_\alpha = \{J_{\alpha_1}, ..., J_{\alpha_q}, ...\}$ and $P_\beta$ processes jobs in $H_\beta = \{J_{j*}, ...\}$. Consider now the subinstance $I'$ of $I$ defined by the set of jobs $J* = J \setminus (H_\alpha \cup H_\beta)$. For the previous theorem there exists a criterion $C*$ such that $a(I') = a(I) - 2$. However $\overline{z}(I') \leq \overline{z}(I) - 1$; thus $\overline{z}(I') \leq k$ and, by induction hypothesis, $a(I') < 2k$ so $a(I) < 2k + 2 = 2\overline{z}(I)$.

Consider the family of instances $I$ with $L = 2m$; $n = 3m$; $r_j = (j+1)/2$ and $d_j = r_j + 1$ for $j = 1, 3, ...2m-1$; $r_j = d_{j-1}$ and $d_j = r_j + L/2$ for $j = 2, 4, ...2m$; $r_j = d_{2(j-2m)}$ and $d_j = r_j + L/2$ for $j = 2m+1, ..., n$. For these instances the optimal solution assigns the $m$ couples $(J_2, J_{2m+1}), (J_4, J_{2m+2}), ..., (J_{2m}, J_n)$ to $m$ machines and the jobs $J_1, J_3, ..., J_{2m-1}$ to another machine. GREEDY uses $m$ machines, independently from the choice of $C$, to process couples $(J_1, J_2), (J_3, J_4), ..., (J_{2m-1}, J_{2m})$ and $m$ more machines, one for each remaining job. Thus $z(I) = m + 1$, $a(I) = 2m$ and the worst case ratio $\frac{a(I)}{z(I)}$ goes to 2 when $m$ grows. □

The authors in [6] introduce another algorithm for which they assume that the set of unassigned jobs is made up of $J_1, J_2, ..., J_u$ and that it is ordered in non decreasing order of $r_j$. Define an oriented graph by:

- a vertex $v_1$ for $J_1$;

- a vertex $v_k$ for each unassigned job $J_k$ with $d_k \leq r_1 + L$;
- an edge $(v_i, v_j)$ of lenght $d_j - r_j$ for each couple $(J_i, J_j)$ such that $r_j \geq d_i$;
- a vertex $v_{u+1}$ and all the edges$(v_i, v_{u+1})$ of 0 lenght.

A path of maximum lenght $v_1$ a $v_{u+1}$ corresponds to a locally optimal assignment. This graph is acyclic and such a path can be constructed in $O(n^2)$ for a total complexity of $O(n^3)$. We can also avoid the explicit definition of the graph: it is possible to scan the couples $(r_j, dj)$ until we find the first $d_j$ for which $d_j > r_1 + L$. Let $t$ be a time in our timeline and for each job $J_j$ with $r_j < t$:

- $M_j$ the lenght of the longest path from $v_1$ to $v_j$;
- $p_j$ the predecessor of $J_j$ in that path;
- $i^* = \arg\max\{M_j : d_j < t\}$;
- $M^* = M_{i^*}$ i.e. the lenght of the longest path finishing before $t$;

For every iteration if $t$ corresponds to some $r_j$ we define $M_j$ by $M^* + (d_j - r_j)$; if $t$ corresponds to some $d_l$, put $M^* = \max\{M^*, M_l\}$. The required assignment is provided via backtracking from $i^*$ through $p_j$.

The authors in [6] solve instances of three types. They consider a discrete timeline $[1, 200]$ to simulate a 20 hours day activity.

- $r_j$ generated with uniform distribution in $[1, 200]$ per ogni $J_j$;
- the 30% of $r_j$ randomly generated with uniform distribution in $[30, 40]$, the 30% in $[130, 140]$ and the remaining 40% in $[1, 29]$, $[41, 129]$ e $[141, 200]$;
- the 20% of $r_j$ randomly generated with uniform distribution in $[30, 40]$, the 20% in $[80, 90]$, the 20% in $[140, 150]$ and the remaining 40% in $[1, 29]$,$[41, 79]$,$[91, 139]$ and $[151, 200]$.

The second and the third scenarios represent real world situations that occur in rail and road transport systems. In every scenario in [6] the authors put $L = 100$. For each type of instance three types of generation are considered for the duration of the jobs:

- $d_j$ randomly generated with uniform distribution in $[r_j + 5, r_j + 10]$;
- $d_j$ randomly generated with uniform distribution in $[r_j + 5, r_j + 20]$;
- $d_j$ randomly generated with uniform distribution in $[r_j + 5, r_j + 40]$.

For each one of the resulting 9 scenarios 5 values for $n$ are considered:50, 100, 200, 500, 1000. Computation experiments suggest that the algorithms in [4] e [6] produce the best results in the second and in the third scenarios. Moreover they show that there is no sensible difference between the second and the third scenario. The processing time grows with the lenght of the jobs.

## 2.3 The Tactical Fixed Job Scheduling

In [7] and [8] authors introduce the Tactical Fixed Job Scheduling (TFJS), another variant of the basic FJS. The aim is to minimize the number of non identical machines needed to process $n$ jobs with fixed release time $r_j$ and deadline $d_j$. Preemption is not allowed. The jobs are divided into groups (called "classes" in [8]); the group $J_j$ belongs to is denoted by $a_j$. The number of job groups is denoted by $g$. Each machine is only able to process jobs from a given subset of job groups. The number of machine classes is denoted by $c$. Let $Q_i$ be the set of job groups that machines of class $i$ can execute and let $K^i$ be the set of all jobs that machines of class $i$ can execute. Authors in [8] also assume that no class is embedded: for

each two classes $i$, $i'$ it never happens that $A_i \subset A_{i'}$. The set $\Gamma_a$ contains all the machines that can process jobs from job group $a$. Let $\Theta$ be the set of all the release times and $\Theta_i$ be the set of all the release times of the jobs that machines of class $i$ can process. Denote by $\Lambda^t$ and $\Lambda$ the number of jobs active at time $t$ and the maximum among the $\Lambda^t$ with $t \in \Theta$. As in [8] we introduce the decision variables $y_i$ representing the number of machines of class $i$ and the binary variables $x_{ij}$ assuming value 1 if $J_j$ is processed by a machine of class $i$ and value 0 otherwise. So we can formulate TFJS as follows:

$$z_{IP} = \min \sum_{i=1}^{c} y_i \tag{2.7}$$

s. t.

$$\sum_{\{J_j : r_j \leq t < d_j \wedge J_j \in K^i\}} x_{ij} \leq y_i \quad i = 1, ..., c \tag{2.8}$$

$$\sum_{i \in \Gamma_a} x_{ij} = 1 \quad \forall J_j \tag{2.9}$$

$$x_{ij} \in \{0, 1\} \tag{2.10}$$

$$y_i \in \{0, 1, ...\} \tag{2.11}$$

The objective function (2.7) aims to minimize the number of machines needed to process all jobs. Constraints (2.8) assures that preemption is not allowed and that the number of jobs executed in parallel by the machines of class $i$ is not greater than $y_i$. Constraints (2.9) makes a job processed only once. The problem is NP-hard (as proved by polynomial reduction to the tridimensional matching) also if preemption is allowed.

A special case of TFJS is the one in which $g = 3$, $c = 2$ and a machine of class $i$ can only process jobs from job groups $i$ and 3. In that special case the problem can be solved to optimality in polynomial time by the following procedure:

**PROCEDURE (ROUND-OFF)**:

Let $z_{LP}$ be the optimal value of the linear relaxation of TFJS;

**if** $z_{LP}$ is integer, then the optimal solution has been found;

**else** solve the extended linear formulation obtained adding constraints $y_1 + y_2 = \lceil z_{LP} \rceil$ to the linear relaxation.

Authors in [8] introduce some lower bounds for $z_{IP}$:

- the optimal value obtained removing the class constraints, namely the basic Fixed Job Scheduling;
- the optimal value of the preemptive relaxation of the problem;
- the optimal value of the linear relaxation obtained relaxing the integrality constraints;
- the optimal value of the lagrangian relaxation of the problem related to the second set of constraints.

In the last case we get a problem with the following criterion function:

$$z_{LR}(v) = \min \sum_{i=1}^{c} y_i + \sum_{j=1}^{n} v_j (1 - \sum_{i \in C_{a_i}} x_{ij}) = \min \sum_{i=1}^{c} (y_i - \sum_{\{J_j : J_j \in K_i\}} v_j x_{ij}) + \sum_{j=1}^{n} v_j$$

where $v_j$ is the multiplicator related to the job $J_j$. This relaxation divides the problem in $c$ subproblems, each one of those corresponds to a machine class.

The authors in [8] define a graph $G_i$ as the graph with nodes $N_i = \{r_1, r_2, ..., r_n, d_1, d_2, ..., d_n\}$ and edges $K_i$: every job in $K_i$ can be seen as an oriented edge of capacity 1 from the vertex corresponding to its release time to the one corresponding to its deadline. $N_i$ will be also denoted by $\{n_{ir} : r = 1, ..., p_i\}$ where $p_i = |N_i|$. $G_i$ will also contain edges of the form $(n_{ir-1}, n_{ir})$ for each $r = 2, ..., p_i$. TFJS can be seen as

a flow problem on the graph $G$ obtained as union of the $G_i$s. Every subproblem rising from the lagrangian relaxation corresponds to a graph $G_i(v)$ obtained from $G_i$ changing the cost of the edge corresponding to $J_j$ to $-v_j$. $z_{LR}$ can be computed via the following procedure:

**PROCEDURE**

**for** $i = 1, ..., c$ **do**

$f := 0; z_i^* := 0$; OPT=false;

**repeat**

$f := f + 1$;

find a minimum cost flow of $f$ units from $n_{i1}$ to $n_{ip_i}$ in $g_i(v)$. Let $z_i$ be the obtained solution;

**if** $z_i + 1 < z_i^*$, then $z_i^* := z_i$

**else** OPT:=true;

repeat until it reaches optimality, $y_i := f - 1$.

$z_{LR}(v)$ is computed as

$$\sum_{i=1}^{c}(z_i^* + y_i) + \sum_{j=1}^{n} v_j.$$

Since $\Lambda_i \leq n$, $y_i \leq \Lambda_i$, that bound can be computed in polynomial time. The authors in [8] implemented the following breaking criterion: the procedure stops when the multiplicators are updated $n$ times or when the difference between the lagrangian lower bound and the following greedy upper bound greedy is less than 1. This latter condition implies that we have found an optimal solution. Computational experiments show that $z_{LR} < \max_v z_{LR}(v)$, despite convergence problems.

Every feasible solution of TFJS corresponds to an upper bound for $z_{IP}$. In [8] authors presented two upper bound: the class covering bound and the greedy bound.

*Class Covering Bound.* Let $\alpha$ be a set of job groups; $\alpha$ is said to be *admissible* if $\alpha \subset Q_i$ for some machine class $i$. Let $\{\alpha_1, ..., \alpha_\nu\}$ be a collection of admissible subsets covering all job groups, i.e.

$$\bigcup_{k=1}^{\nu} \alpha_k = \{1, ..., g\}.$$

An admissible subset $\alpha'$ will be said to be *non redundant* in such a cover if $\{\alpha_1, ..., \alpha_\nu\} \setminus \{\alpha'\}$ does not cover all job groups anymore.

**Theorem 7** ([8]). *If $\{\alpha_1, ..., \alpha_\nu\}$ is a collection of admissible subsets covering all job groups, then*

$$\sum_{k=1}^{\nu} \Lambda_{\alpha_k}$$

*is an upper bound for $z_{IP}$. If all subsets in the cover are not redundant, then that bound is an approximated solution for TFJS.*

*Proof.* If $\alpha_k \subset Q_i$ is an admissible subset, then all jobs $J_j$ with $a_j \in \alpha_k$ can be processed by $\Lambda_{\alpha_k}$ machines of class $i$. Hence, if $\{\alpha_1, ..., \alpha_\nu\}$ is a collection of admissible subsets covering all job groups, then all jobs can be processed by $\sum_{k=1}^{\nu} \Lambda_{\alpha_k}$ machines.

The second part of the statement is proved by the following family of instances. Every subset $\alpha_k$ contains at least one job group which is not in the other elements of the cover: denote by $a(k)$ the first job group with that property. Consider now an instance with the following structure: divide the timeline in $\nu$ subintervals, the $k-$th subinterval will only contain jobs from $a(k)$. For this instance we have that

$$z_{IP} = \sum_{k=1}^{\nu} \Lambda_{\alpha_k}.$$

□

To obtain a cover without redundant subsets we can procede as follows. Let $\{\alpha_1, ..., \alpha_S\}$ be a list of all admissible subsets (note that $S = O(c2^g)$) and introduce the decision variables $x_s$ $(s = 1, ..., S)$ assuming value 1 if $\alpha_s$ is chosen and value 0 otherwise. So we have the following model:

$z_{CC} = \min \sum_{s=1}^{S} \Lambda_{\alpha_s} x_s$

s. t.

$\sum_{\{s:a\in\alpha_s\}} x_s = 1 \quad a = 1, ..., g$

$x_s \in \{0, 1\}$.

If $c$ and $g$ are fixed we can compute $z_{CC}$ by enumeration in a number of operations that is independent from the number of jobs.

We can obtain a cover of admissible subsets taking the singletons of the form $\{\alpha_k\}$. In this case we get the lower bound

$$\sum_{a=1}^{g} \Lambda_a$$

in $O(n \log n)$.

*Greedy upper bound.* The authors in [8] propose an upper bound computed with an heuristic algorithm. This procedure increases the number of machines opened to build a feasible solution. We can sum up an iteration of the heuristic algorithm as follows:

HEURISTIC GREEDYPROC

$K := \{J_1, ..., J_n\}$;

$y_v := 0 \quad \forall i$;

repeat

search for the locally optimal class $i^*$;

$y_{i^*} := y_{i^*} + 1$;

$K := K \setminus W^{i^*}$ = set of all jobs that can be processed from a new machine in $i^*$.

repeat until $K = \emptyset$.

Suppose that during the execution we have $K \neq \emptyset$: this implies that we don't have enough machines to process all jobs. Increase by 1 the number of machines used for every class and obtain $i^*$ as the class for which this increment is more "suitable". This advantage is computed has the total (lagrangian) value of the new jobs that could be processed.

For the computational experiments in [8] the time horizon limit is set to 1000 and the following parameters are set:

- Number of job groups: $g = 4$, $g = 5$ or $g = 6$;
- Number of jobs: $n = 100$, $n = 200$ or $n = 300$;
- Maximum job duration $D$: $D = 100$, $D = 200$ or $D = 300$;
- Job characteristics: The job group to which a job $J_j$ belongs to is chosen randomly in $\{1, ..., g\}$, duration $t_j = (d_j - r_j)$ of a job randomly chosen with uniform distribution in $(0, D]$, $r_j$ randomly chosen in $(0, T - t_j]$.
- Machine classes: two cases are studied. In the first case every machine can process jobs from two job groups and the number of classes is chosen so that every combination of two job groups is considered; in the second case only a few combinations are considered. For $g = 4$, $g = 5$ e $g = 6$ the authors in [8] put $c = 3$, $c = 4$, $c = 5$. The set $Q_i$ is made so that $Q_i = \{i, i + 1\}$ per $i = 1, ..., g - 1$.

## 2.4 The Tactical Fixed Job Scheduling with Spread Time Constraints

An istance of TFJSS consists of $n$ jobs $J_j$ $(j = 1, ..., n)$ that must be processed without preemption from a fixed release time (or starting time) $r_j$ to a fixed deadline $d_j$ on $m$ non identical machines that can process only one job at a time $(m < n)$. Each machine can only work for a fixed number $L$ of consecutive time units: as already mentioned, the spread-time is defined as the range from the first job and the last job assigned to a machine. Moreover machines are divided into $c$ classes: machines belonging to a class can only process jobs from a given subset of jobs. The goal is to minimize the number of machines required to process all jobs.

Let $M^i$ be the set of machines of class $i$ $(1 \leq i \leq c)$ and $C_j$ $(1 \leq j \leq n)$ be the set of classes containing all the machines able to process $J_j$. Let $K^i$ be the set of jobs that can be processed from the machines of $M^i$. Authors in [9] also introduce a weight $w_i$ for the machines of class $i$, moreover they suppose the jobs ordered in non decreasing order of $r_j$. They present two formulation for TFJSS: an integer programming model and a set covering model for a column generation procedure to be embedded in a branch-and-price algorithm. Two jobs $J_j$ and $J_k$ will be said to be *compatible* if they can be processed by the same machine. For each $J_j \in K^i$ $(1 \leq i \leq c)$ let $A_j = \{J_k \in K^i : r_k \leq r_j \leq d_k \vee r_j \leq r_k \leq d_j \vee d_k - r_j > L \vee d_j - r_k > L\}$ be the set of jobs that are not compatible with $J_j$. We define the decision variables $y_k^i$, that are equal to 1 if machine $k \in M^i$ is used to perform at least one job and equal to 0 otherwise, and the decision variables $x_{jk}^i$ that assume value 1 if job $J_j \in K^i$ is assigned to machine $k \in M^i$ and value 0 otherwise. So we can model the TFJSS as follows:

$$z = \min \sum_{i=1}^{c} w_i \sum_{k \in M^i} y_k^i \tag{2.12}$$

s. t.

$$x_{jk}^i \leq y_k^i \quad J_j \in K^i, k \in M^i, i = 1, ..., c \tag{2.13}$$

$$x_{jk}^i + x_{lk}^i \leq 1 \quad J_l \in A_j, J_j \in K^i, k \in M^i, i = 1, ..., c \tag{2.14}$$

$$\sum_{i \in C_j} \sum_{k \in M^i} x_{jk}^i = 1 \quad j = 1, ..., n \tag{2.15}$$

$$x_{jk}^i, y_k^i \in \{0, 1\} \quad J_j \in K^i, k \in M^i, i = 1, ..., c. \tag{2.16}$$

The objective functions (2.12) requires the minimization of the cost (weight) of machines needed to perform all jobs. Constraints (2.13) assure that a machine is used only when at least one job is assigned to it. Constraints (2.14) assure that the compatibility relations are respected. Constraints (2.15) make a job executed once and by a unique machine.

For implementing their Column Generation procedure the authors in [9] present a Set Covering model for TFJSS. Every set of compatible jobs (that can be processed by a unique machine) will be called "single-machine schedule" (column). For every $i = 1, ..., c$ they define the binary constants $a_{js}^i$ with value 1 if and only if $J_j \in K^i$ is in column $s$ and value 0 otherwise. The vector $a_s^i = (a_{1s}^i, ..., a_{ns}^i)^T$ detects the jobs of column $s$ that can be processed by a machine in $M^i$. Let $S^i$ be the set of all columns containing jobs of $K^i$. Now introduce the binary decision variables $x_s^i$ assuming value 1 if and only if column $s$ is taken in the solution and value 0 otherwise.

$$z_I = \min \sum_{i=1}^{c} \sum_{s \in S^i} w_i x_s^i \tag{2.17}$$

s. t.

$$\sum_{i=1}^{c} \sum_{s \in S^i} a_{js}^i x_s^i \geq 1 \quad j = 1, ..., n \tag{2.18}$$

$$x_s^i \in \{0, 1\} \quad s \in S^i, i = 1, ..., c \tag{2.19}$$

The objective function (2.17) aims to minimize the cost of the selected machine. Constraints (2.18) assure that every job is in at least a column; moreover the corresponding dual variables will be not negative.

In the algorithm in [9] constraints (2.19) are relaxed to obtain a new formulation RP from initial model, call it P: denote by $X_{RP}$ and $z_{RP}$ the optimal solution of RP and its corresponding value respectively.

The first step in a Column Generation procedure is the detection of a restricted subset of columns: this subset describes the restricted problem that will be solved to optimality. In [9] new columns are added if necessary via a dynamic programming procedure. The choice of the initial restricted set of columns is provided by a procedure similar to the greedy procedure described in [6].

**ALGORITMH GR** ([9])

$S = \emptyset$, $j := 1$, order the jobs in non decreasing order of $r_j$.

if $j > n$, STOP, output a feasible solution for the relaxation RP corresponding to the set of columns $S$.

if there exists a column $s \in S$ to which $J_j$ can be added to form a new column $s'$, remove $s$ from $S$ and put $s'$ in $S$;

else, let $h = \min\{i : i \in C_j\}$, construct a new column $s^h$ including only $J_j$ eand add it to $S$.

$j = j + 1$ and go back to the first **if**.

The reduced costs associated with the (relaxed) variables $x_s^i$ are defined by:

$$P_s^i = w_i - \sum_{J_j \in K^i} \lambda_j a_{js}^i,$$

where $\lambda_1, ..., \lambda_n$ are the values of the dual variables corresponding to the solution of the relaxed problem RP. For a well known result of the duality theory a solution to that problem is optimal if the reduced costs corresponding to the variables are non negative: thus to see if a solution is optimal we should scan the reduced costs searching for negative values. We want to detect the minimum among those values or, since $w_i$ is constant for every $s \in S^i$, the maximum among the values

$$\tilde{P}_s^i := \sum_{J_j \in K^i} \lambda_j a_{js}^i.$$

If $\tilde{P}_s^i \leq w_i \; \forall i$, we have found an optimal solution for the relaxed problem; otherwise we need to add more columns. Authors in [9] introduce a "pricing" procedure based on dynamic programming and on the fact that machines processe jobs in non decreasing order of $r_j$. This procedure is combined with a branch-and-bound algorithm that grants the integrality of the solution.

A successor of a job $J_j$ is a job that immediately follows $J_j$ in at least a column; Let $K_j^+$ be the set of all successors of $J_j$.The precedence restrictions are indicated by

$$\wp = \bigcup_{j=1}^{n} K_j^+.$$

$\forall 1 \leq j \leq k \leq n$ and $i = 1, ..., c$ let $F_i(j,k)$ be the set ol all columns in $S^i$ in which $J_j$ eand $J_k$ are the first and the last job respectively, and let $f_i(j,k) = \max_{s \in F_i(j,k)} \tilde{P}_s^i$ if $F_i(j,k) \neq \emptyset$, 0 otherwise. After posing $f_i(j,j) = \lambda_j \; \forall J_j \in K^i \; (i =, ..., c)$, the ricorsion scheme for $k = j+1, ..., n$ and $J_k \in K^i$ is:

- $f_i(j,k) = 0$ if $J_k \in A_j$

- $f_i(j,k) = \max_{l:j\leq l<k; J_k \in K_l^+} f_i(j,l) + \lambda_k$ if $J_k \notin A_j$

Let $f_i^* = \max_{1\leq j\leq k\leq n} f_i(j,k)$ $(i =, ..., c)$: if $f_i^* \leq w_i$ $\forall i$ the current solution is optimal; else, it is necessary to add more columns to the restricted problem. Those columns will be selected among the ones for which $f_i^* > w_i$. Greater is the number of columns added for each iteration, then smaller is the number of iterations needed, however the problem to be solved optimally will be heavier. One can choose the columns determining the maximum value of $f_i^*$. The computational complexity of the procedure is $O(cn^2)$.

The concepts of predecessor and successor are the key for the branch-and-bound algorithm that is combined with the previous procedure. The jobs of a column $s$ are assigned to a machine in a fixed order, namely in non decreasing order of $r_j$: every job of $s$ will have one predecessor and one successor that are uniquely defined by the column, apart from the first and the last ones for which we define slack jobs $J_0$ and $J_{n+1}$ representing the predecessor of the first job and the successor of the last job in the column. If $X_{RP} = \{\tilde{x}_s^i : s \in S^i, i = 1, ..., c\}$ is an optimal solution for RP, let $\Phi(X_{RP})$ be the set of all columns corresponding to the fractional components of $X_{RP}$. Hence $\Phi(X_{RP}) = \emptyset$ if and only if $X_{RP}$ is integer. Moreover let $S_j(X_{RP}) \subseteq \Phi(X_{RP})$ be the of all columns containing $J_j$, and let $K_f(X_{RP})$ be the set of all jobs corresponding to fractional values in $X_{RP}$ on machines of class $i \in \tilde{I}(s) = \{i : 1 \leq i \leq c, s \in S^i \wedge 0 < \tilde{x}_s^i < 1\}$. A job can be removed from the columns of $\Phi(X_{RP})$ without making the solution worse when it is contained in a column $s$ with $\tilde{x}_s^i = 1$.

The authors in [9] prove two statements justifying the adopted strategy.

**Theorem 8** ([9]). *If every $J_j \in K_f(X_{RP})$ has always the same successor in every column $s \in S_j(X_{RP})$, then there exists an optimal solution$X_{RP}^*$ for RP in which*

*every* $J_j \in K_f(X^*_{RP})$ *has the same predecessor and the same successor in every* $s \in S_j(X^*_{RP})$.

*Proof.* Let $J_l \in K_f(X_{RP})$ be the job with the greater index $l$, clearly it is the last job in every column of $S_l(X_{RP})$ and it has always the same successor $J_{n+1}$. Suppose now that $J_l$ has at least two different predessors in different columns of $S_l(X_{RP})$ and that $J_k \neq J_0$ is one of these. Every column in $\Phi(X_{RP})$ containing $J_k$ must also contain $J_l$ by hypothesis. Note that for every $J_j \in K_f(X_{RP})$, if $\Phi(X_{RP}) \neq \emptyset$ we have

$$\sum_{s \in S_j(X_{RP})} \sum_{i \in \tilde{I}(s)} x_s^i \geq 1.$$

Since $S_k(X_{RP}) \subseteq S_l(X_{RP})$, substituting $j$ with $k$ in the previous inequality, we can remove $J_l$ from every column in $S_l(X_{RP}) \setminus S_k(X_{RP})$; so we get a new optimal (LP) solution $X'_{RP}$ such that $\Phi(X'_{RP}) \subseteq \Phi(X_{RP})$. Note that$X'_{RP}$ satisfies our hypothesis and that $J_l$ has the same predecessor in every column of $S_l(X'_{RP})$. This process can be iterated substituting $X_{RP}$ with $X'_{RP}$ until the optimal solution $X^*_{RP}$ is found. □

**Theorem 9** ([9])**.** *If every* $J_j \in K_f(X_{RP})$ *has the same predecessor and successor in every* $s \in S_j(X_{RP})$*, then it is possible to find a feasible, and then optimal solution.*

*Proof.* $S_j(X_{RP})$ contains exactly one column for each $J_j \in K_f(X_{RP})$ by hypothesis. Thus the sets of jobs corresponding to the columns of $\Phi(X_{RP})$ are pairwise disjoint. Once fixed a column $s \in \Phi(X_{RP})$, for the previous theorem $\tilde{I}(s)$ contains at least two elements becasuse of a fractional value of $x_s^i$ for some $i \in \tilde{I}(s)$. The corresponding weights $\{w_i : i \in \tilde{I}(s)\}$ are all equal, if they were not it would be possible

to build a new solution in proving the value of the criterion function. So, fixing $i \in \tilde{I}(s)$, assigning in $X_{RP}$ value 1 to $x_s^i$ and value 0 to $x_s^{i'}$ for every $i' \in \tilde{I}(s) \setminus \{i\}$, we find a new integer solution.The process described above can be iterated for every column $s \in \Phi(X_{RP})$ until an integer optimal solution is found. □

We say a job $J_j \in K_f(X_{RP})$ is separated in the current solution if it has more than one successor in different $S_j(X_{RP})$. Authors in [9] design their branch-and-price tree such that at each node of the tree, they find a separated job with lowest index, and then create descendant nodes in such a way that each of descendant nodes corresponds to a fixed successor. Consequently, the branching strategy eventually eliminates all separated jobs, which is a sufficient condition for an integer solution.

In their experiments the authors in [9] showed that a breadth-first approach for the branching tree is better than a depth-first: the neighbour nodes are explored before descending in the tree.

The authors in [9] take into account only instances with at most 300 jobs for their computational experiments. They use some of the parametres used in [8] and they fix a limit of one hour for the time given to the algorithm for finding the optimal solution: The instances for which this limit is met are considered "not solved". The algorithm in [9] is showed to be more efficient than CPLEX: this latter cannot solve the instances in the time limit (formulation ILP). The instances are generated as follows:

- Number of jobs: $n = 100$, $n = 200$ or $n = 300$;
- Number of classes: $c = 2^g - 1$, where $g$ is the number of job groups, $g = 3$, $g = 4$ or $g = 5$;

- Job duration: generated with uniform distribution in $[1, 100]$, $[41, 100]$ or $[81, 100]$
- Spread-time: $L = 300$, $L = 400$ or $L = 500$.

For each one of the 81 scenarios they generate 10 instances that have been tested both with the branch-and-price algotithm and with CPLEX; however the numbers in the tables are only the averages of the times (running time) of the branch-and-price algorithm because, as already mentioned, CPLEX cannot solve the instances within the time limit. The number in superscript represent the number of instances that are not solved within the time limit.

Table 1: execution time (in seconds) with $n = 100$

| Spread $L$ | Job Duration | 3 Job Groups | 4 Job Groups | 5 Job Groups |
|---|---|---|---|---|
| 300 | $1-100$ | 2.31 | 4.09 | 2.41 |
| | $41-100$ | 0.43 | 0.68 | 0.44 |
| | $81-100$ | 0.16 | 0.16 | 0.29 |
| 400 | $1-100$ | 14.29 | 7.72 | 10.84 |
| | $41-100$ | 2.41 | 2.06 | 4.32 |
| | $81-100$ | 0.66 | 0.95 | 2.17 |
| 500 | $1-100$ | 37.12 | 20.13 | 23.63 |
| | $41-100$ | 13.70 | 6.52 | 12.07 |
| | $81-100$ | 1.86 | 4.23 | 5.63 |

Table 2: execution time (in seconds with $n = 200$

| Spread $L$ | Job Duration | 3 Job Groups | 4 Job Groups | 5 Job Groups |
|---|---|---|---|---|
| 300 | $1-100$ | 26.30 | 37.68 | 41.73 |
| | $41-100$ | 3.29 | 9.98 | 12.02 |
| | $81-100$ | 0.65 | 1.14 | 1.98 |
| 400 | $1-100$ | 92.49 | 115.04 | 328.58 |
| | $41-100$ | 273.07 | 98.77 | 24.81 |
| | $81-100$ | 7.88 | 12.16 | 16.57 |
| 500 | $1-100$ | 325.25 | 595.69 | 699.14 |
| | $41-100$ | 193.60 | 170.32 | 204.95 |
| | $81-100$ | 16.24 | 31.93 | 48.95 |

Table 3: execution time (in seconds) with $n = 300$

| Spread $L$ | Job Duration | $n = 100$ | $n = 200$ | $n = 300$ |
|---|---|---|---|---|
| 300 | $1 - 100$ | 14.25 / 0.00 | 16.02 / 0.00 | 12.69 / 1.50 |
| | $41 - 100$ | 8.48 / 0.00 | 11.39 / 0.00 | 10.67 / 0.00 |
| | $81 - 100$ | 6.02 / 0.00 | 9.36 / 0.00 | 10.74 / 0.00 |
| 400 | $1 - 100$ | 14.74 / 0.00 | 14.65 / 0.00 | 11.06 / 5.15 |
| | $41 - 100$ | 10.29 / 0.00 | 11.56 / 0.00 | 6.94 / 1.08 |
| | $81 - 100$ | 11.75 / 0.00 | 10.43 / 0.00 | 12.42 / 0.00 |
| 500 | $1 - 100$ | 17.34 / 0.00 | 14.83 / 0.00 | 12.01 / 10.43 |
| | $41 - 100$ | 13.14 / 0.00 | 10.52 / 0.00 | 10.66 / 4.11 |
| | $81 - 100$ | 10.88 / 0.00 | 8.72 / 0.00 | 9.37 / 0.00 |

In the tables 1-3, the running time for solving each case increases with the spread-time and with the range of job duration.

The authors in [9] also study how the solution changes with the flexibility of the machines: that is defined as capacity of the machines to process jobs from different job groups. The parameters are almost the same of the previous experiments apart from the number of job groups and of jobs that are decreased to 10 and 50 respectively. The total number of machine classes is $c = 2^{10} - 1$ with 10 subsets of classes where each one of the $C_i^{10} = 10!/(i!(10-i)!)$ $i-$tier contains all the machines that can process from $i$ job groups. Let $\tilde{M}^i$ be the set of all the machines of the classes of $i-$tier. Greater is yhe value of $i$, more flexible the machines of $\tilde{M}^i$ are. One of the detected data is the average number of used machines. The autgors in [9] first consider two extreme cases (Tables 4-5): in the first one (total flexibility) the weights are all equal to 1 for every $\tilde{M}^i$; in the second one (no flexibility) the weights are equal to 1 for $\tilde{M}^1$ and 999 for the $\tilde{M}^i$ with $i \geq 2$. For each one of the described scenarios 10 instances are generated; moreover the percentage of the machine used related to $\tilde{M}^i$ are indicated. From tables 4 and 5 it is clear, as one might suppose, that in the case of total flexibility the number of machines use is lower than the other case.

Table 4: total flexibility

| Spread $L$ | Job Duration | Machine Distributions in Percentage | | | | | | | | | | Avg# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | $1-100$ | 15.7 | 11.0 | 3.3 | 7.9 | 24.3 | 19.1 | 10.8 | 7.2 | 0.6 | 0.0 | 16.4 |
| | $41-100$ | 30.0 | 8.5 | 2.9 | 7.1 | 28.4 | 15.8 | 4.6 | 2.1 | 0.6 | 0.0 | 19.7 |
| | $81-100$ | 31.0 | 37.4 | 2.7 | 4.9 | 17.0 | 5.7 | 0.9 | 0.5 | 0.0 | 0.0 | 23.0 |
| 400 | $1-100$ | 9.9 | 8.0 | 2.4 | 2.3 | 22.4 | 19.7 | 20.8 | 10.1 | 4.5 | 0.0 | 13.6 |
| | $41-100$ | 16.2 | 11.0 | 2.1 | 9.6 | 24.7 | 21.6 | 9.7 | 3.8 | 0.7 | 0.6 | 16.3 |
| | $81-100$ | 21.1 | 9.2 | 1.6 | 9.4 | 31.7 | 14.9 | 6.5 | 5.4 | 0.0 | 0.0 | 18.4 |
| 500 | $1-100$ | 12.1 | 3.2 | 1.0 | 3.3 | 14.8 | 20.8 | 19.9 | 10.4 | 11.2 | 3.3 | 11.8 |
| | $41-100$ | 16.6 | 4.5 | 1.3 | 7.9 | 19.1 | 23.0 | 15.9 | 8.2 | 3.7 | 0.0 | 14.2 |
| | $81-100$ | 18.1 | 5.7 | 0.6 | 9.8 | 21.9 | 26.4 | 9.5 | 6.8 | 1.3 | 0.0 | 15.8 |

Table 5: no flexibility

| Spread $L$ | Job Duration | Machine Distributions in Percentage | | | | | | | | | | Avg# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | $1-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 31.7 |
| | $41-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 34.0 |
| | $81-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 36.5 |
| 400 | $1-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 27.8 |
| | $41-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 30.4 |
| | $81-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 31.4 |
| 500 | $1-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 24.1 |
| | $41-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 27.2 |
| | $81-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 29.5 |

The authors in [9] gradually increase the flexibility of the machines studying some intermediate scenarios: the numbers in tables 6 and 7 are not so different from the ones reported in table 4; this observation leads to the conclusion that total flexibility does not provide massive improvements (reduction of the number of machines used) with respect to the case in which the weights equal to 1 are the ones in $\tilde{M}^1$,$\tilde{M}^2$ ed $\tilde{M}^3$.

Table 6: reduced flexibility (pweights equal to (1,1,999,...,999))

| Spread $L$ | Job Duration | Machine Distributions in Percentage | | | | | | | | | | Avg# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | $1-100$ | 6.0 | 94.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 17.9 |
| | $41-100$ | 9.6 | 90.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 20.6 |
| | $81-100$ | 27.6 | 72.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 23.8 |
| 400 | $1-100$ | 2.0 | 98.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15.1 |
| | $41-100$ | 5.0 | 95.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 17.0 |
| | $81-100$ | 9.1 | 90.9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 18.6 |
| 500 | $1-100$ | 0.8 | 99.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 13.2 |
| | $41-100$ | 1.8 | 98.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15.2 |
| | $81-100$ | 5.3 | 94.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 16.6 |

Tabella 7: little flexibility (weights equal to (1,1,1,999,...,999))

| Spread $L$ | Job Duration | Machine Distributions in Percentage | | | | | | | | | | Avg# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | $1-100$ | 12.8 | 7.8 | 79.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 16.5 |
| | $41-100$ | 24.3 | 23.6 | 52.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 19.7 |
| | $81-100$ | 31.3 | 38.3 | 30.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 23.0 |
| 400 | $1-100$ | 4.0 | 4.8 | 91.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 13.6 |
| | $41-100$ | 14.4 | 6.0 | 79.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 16.3 |
| | $81-100$ | 16.4 | 11.5 | 72.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 18.4 |
| 500 | $1-100$ | 4.7 | 0.8 | 94.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 11.8 |
| | $41-100$ | 11.7 | 5.4 | 83.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 14.2 |
| | $81-100$ | 11.2 | 13.8 | 75.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15.8 |

Finally the authors in [9] study some cases of more general weights. The weights are put equal to 1 for the machines that can process jobs only from a unique job group and equal to $1+kp$ for the machines that can process jobs from $k$ job groups more. The parameter $p$ is set to 0,03, 0,20, 0,80 o 1,00 (Tables 8-11). For the first three values of $p$ computational experiments show that the machines that can process jobs from 3 or 4 job groups are the best choice in the majority of cases. In the case $p = 1$ no evident improvement has been recorded.

Table 8: case of weights equal to $1 + kp$ with $p = 0,03$

| Spread $L$ | Job Duration | Machine Distributions in Percentage | | | | | | | | | | Avg# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | $1-100$ | 25.6 | 47.5 | 23.7 | 3.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 16.4 |
| | $41-100$ | 39.3 | 38.7 | 21.1 | 0.9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 19.7 |
| | $81-100$ | 42.8 | 42.2 | 15.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 23.0 |
| 400 | $1-100$ | 21.4 | 44.3 | 31.2 | 3.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 13.6 |
| | $41-100$ | 30.4 | 44.0 | 24.4 | 1.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 16.3 |
| | $81-100$ | 37.0 | 47.0 | 15.5 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 18.4 |
| 500 | $1-100$ | 22.4 | 44.0 | 30.9 | 2.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 11.8 |
| | $41-100$ | 27.6 | 45.0 | 25.4 | 1.9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 14.2 |
| | $81-100$ | 27.7 | 50.7 | 19.6 | 2.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15.8 |

Tabella 9: case of weights equal to $1 + kp$ with $p = 0,20$

| Spread $L$ | Job Duration | Machine Distributions in Percentage | | | | | | | | | | Avg# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | $1-100$ | 25.8 | 48.7 | 21.1 | 3.7 | 0.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 16.4 |
| | $41-100$ | 36.9 | 43.5 | 18.5 | 1.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 19.7 |
| | $81-100$ | 44.1 | 40.5 | 15.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 23.0 |
| 400 | $1-100$ | 22.1 | 43.5 | 31.4 | 2.2 | 0.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 13.6 |
| | $41-100$ | 26.8 | 51.7 | 20.3 | 0.7 | 0.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 16.3 |
| | $81-100$ | 38.6 | 43.3 | 18.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 18.4 |
| 500 | $1-100$ | 24.9 | 39.8 | 30.8 | 4.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 11.8 |
| | $41-100$ | 25.5 | 51.8 | 20.5 | 1.6 | 0.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 14.3 |
| | $81-100$ | 29.1 | 48.7 | 20.8 | 0.7 | 0.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15.8 |

Tabella 10: case of weights equal to $1 + kp$ with $p = 0,80$

| Spread $L$ | Job Duration | Machine Distributions in Percentage | | | | | | | | | | Avg# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | $1-100$ | 36.8 | 45.6 | 16.3 | 0.6 | 0.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 17.5 |
| | $41-100$ | 53.3 | 34.4 | 10.8 | 1.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 21.3 |
| | $81-100$ | 63.2 | 27.0 | 9.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 25.0 |
| 400 | $1-100$ | 36.7 | 40.1 | 21.9 | 0.7 | 0.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 14.9 |
| | $41-100$ | 40.8 | 43.5 | 15.1 | 0.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 17.4 |
| | $81-100$ | 51.6 | 38.7 | 8.7 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 19.8 |
| 500 | $1-100$ | 31.0 | 49.9 | 16.6 | 2.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12.7 |
| | $41-100$ | 40.9 | 43.2 | 14.9 | 1.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15.5 |
| | $81-100$ | 37.7 | 46.9 | 14.2 | 1.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 16.6 |

Tabella 11: case of weights equal to $1 + kp$ with $p = 1,00$

| Spread $L$ | Job Duration | Machine Distributions in Percentage | | | | | | | | | | Avg# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | $1-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 31.7 |
| | $41-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 34.0 |
| | $81-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 36.5 |
| 400 | $1-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 27.8 |
| | $41-100$ | 99.7 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 30.3 |
| | $81-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 31.4 |
| 500 | $1-100$ | 99.6 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 24.0 |
| | $41-100$ | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 27.2 |
| | $81-100$ | 99.6 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 29.4 |

# Chapter 3

# Original Contribution

## 3.1 Mathematical Formulation

An istance of TFJSS consists of $n$ jobs $J_j$ $(j = 1, ..., n)$ that must be processed without preemption from a fixed release time (or starting time) $r_j$ to a fixed deadline $d_j$ on $m$ non identical machines that can process only one job at a time $(m < n)$. Each machine can only work for a fixed number $L$ of consecutive time units: as already mentioned, the spread-time is defined as the range from the first job and the last job assigned to a machine. Moreover machines are divided into $c$ classes: machines belonging to a class can only process jobs from a given subset of jobs. The goal is to minimize the number of machines required to process all jobs.

Let $M^i$ be the set of machines of class $i$ $(1 \leq i \leq c)$ and $C_j$ $(1 \leq j \leq n)$ be the set of classes containing all the machines able to process $J_j$. Let $K^i$ be the set of jobs that can be processed from the machines of $M^i$. Two jobs $J_j$ and $J_k$ are said to be compatible if they can be performed by the same machine. For each $J_j \in K^i$ $(1 \leq i \leq c)$ let $A_j = \{J_k \in K^i : r_k \leq r_j \leq d_k \vee r_j \leq r_k \leq d_j \vee d_k - r_j > L \vee d_j - r_k > L\}$ be the set of jobs that are not compatible with $J_j$. We define the

decision variables $y_k^i$, that are equal to 1 if machine $k \in M^i$ is used to perform at least one job and equal to 0 otherwise, and the decision variables $x_{jk}^i$ that assume value 1 if job $J_j \in K^i$ is assigned to machine $k \in M^i$ and value 0 otherwise. So we can model the TFJSS as follows:

$$z = \min \sum_{i=1}^{c} \sum_{k \in M^i} y_k^i \tag{3.1}$$

s. t.

$$x_{jk}^i \leq y_k^i \quad J_j \in K^i, k \in M^i, i = 1, ..., c \tag{3.2}$$

$$x_{jk}^i + x_{lk}^i \leq 1 \quad J_l \in A_j, J_j \in K^i, k \in M^i, i = 1, ..., c \tag{3.3}$$

$$\sum_{i \in C_j} \sum_{k \in M^i} x_{jk}^i = 1 \quad j = 1, ..., n \tag{3.4}$$

$$x_{jk}^i, y_k^i \in \{0, 1\} \quad J_j \in K^i, k \in M^i, i = 1, ..., c. \tag{3.5}$$

The objective function (3.1) requires the minimization of the number of machines needed to perform all jobs. Constraints (3.2) assure that a machine is used only when at least one job is assigned to it. Constraints (3.3) assure that the compatibility relations are respected. Constraints (3.4) make a job executed once and by a unique machine.

It is now clear that in this interpretation of the problem there are perfect correspondences between machines and workers and jobs and passengers. Notably the spread-time corresponds to the lenght of a day duty.

## 3.2 Complexity

In this section the intractability of the TFJSS is proved via polynomial reduction to the FJSS ([4], [6]).

**Theorem 10** ([10]). *The TFJSS is NP-complete.*

*Proof.* We introduce the mathematical formulation of the FJSS ([4]). In an instance of this problem it is required to schedule $n$ jobs $J_j$ $(j = 1, ..., n)$ on $m$ identical machines for which there is the same spread-time limit $L$. Jobs have fixed starting time $r_j$ and deadline $d_j$ and they must be processed without preemption. For each $J_j$ let $A_j = \{J_k : r_k \leq r_j \leq d_k \vee r_j \leq r_k \leq d_j \vee d_k - r_j > L \vee d_j - r_k > L\}$ be the set of jobs that are not compatible with $J_j$. We define decision variables $y_k$ that assume value 1 if machine $k$ is used and 0 otherwise; and $x_{jk}$ that assume value 1 if job $J_j$ is processed by machine $k$ and 0 otherwise. We can formulate as follows:

$$z' = \min \sum_{k=1}^{m} y_k \tag{3.6}$$

s. t.

$$x_{jk} \leq y_k \quad \forall J_j, k = 1, ..., m \tag{3.7}$$

$$x_{jk} + x_{lk} \leq 1 \quad J_l \in A_j; \forall J_j; k = 1, ..., m \tag{3.8}$$

$$\sum_{k=1}^{m} x_{jk} = 1 \quad j = 1, ..., n \tag{3.9}$$

$$x_{jk}, y_k \in \{0, 1\} \quad \forall J_j, k = 1, ..., m. \tag{3.10}$$

Putting $C = 1$ in model (3.1)-(3.5) one can produce a polynomial reduction to (3.6)-(3.10); this latter is NP-complete so TFJSS is NP-complete too. $\diamond$ $\square$

## 3.3 Lower Bounds

In this section we introduce some lower bounds for the optimal value of TFJSS. Consider a discrete time-line $[0, T]$, a fixed time $t^0 \in [0, T]$, let $n^0$ be the number

of active jobs at $t^0$ (computable in polynomial time as in [1]). Put $t_1^0 = t^0 + L + 1$, $t_{-1}^0 = t^0 - L - 1$. More generally

$$t_p^0 = t_{p-1}^0 + L + 1,$$

$$t_{-p}^0 = t_{1-p}^0 - L - 1$$

when they exist in $[0, T]$. Denote with $n_p^0$ the number of active jobs at time $t_p^0$ with $p \in Z$.

**Theorem 11** ([10])**.**

$$B_1(t^0) = \sum_p n_p^0$$

*is a lower bound for the optimal value of TFJSS.*

*Proof.* In fact a machine working at time $t_p^0$ cannot be working at time $t_{p+1}^0$ or at time $t_{p-1}^0$. Denote with $z^*$ the optimal value of (1)-(5). Clearly $z^* \geq B_1(t^0)$ because $B_1(t^0)$ does not take into account the jobs that have release time and deadline between some $t_p^0$ and $t_{p+1}^0$. We obtain a family of lower bounds moving $t^0 \in [0, T]$ obtainable in polynomial time.$\diamond$ $\square$

Moreover put

$$\overline{B}_1 = \max_{t^0 \in [0,T]} B_1(t^0).$$

Clearly $\overline{B}_1$ is a lower bound for the optimal value of TFJSS.

Take now $t^0 \in [0, T]$ and put

$$\tau_p^0 = \min\{t : t \geq t_p^0 \wedge n(t) \geq 1\},$$

$$\tau_{-p}^0 = \max\{t : t \leq t_{-p}^0 \wedge n(t) \geq 1\},$$

when they exist in $[0, T]$, where $n(t)$ is the number of active jobs at time $t$. Note that if $n_p^0 \geq 1$, then $\tau_p^0 = t_p^0$.

**Theorem 12** ([10])**.**

$$B_2(t^0) = \sum_p \tau_p^0$$

*is a lower bound for the optimal value of TFJSS.*

*Proof.* In fact a machine working at time $\tau_p^0$ cannot be working at time $\tau_{p+1}^0$ or at time $\tau_{p-1}^0$. Denote with $z^*$ the optimal value of (1)-(5). Clearly $z^* \geq B_2(t^0)$ because $B_2(t^0)$ does not take into account the jobs that have release time and deadline between some $\tau_p^0$ and $\tau_{p+1}^0$. We obtain a family of lower bounds for the optimal value of TFJSS moving $t^0 \in [0, T]$. $\diamond$ □

Moreover

$$\overline{B}_2 = \max_{t^0 \in [0,T]} B_2(t^0)$$

is a lower bound for the optimal value of TFJSS.

## 3.4 An Heuristic Algorithm

### 3.4.1 The Algorithm

In this section an heuristic algorithm based on a greedy approach is presented, nevertheless it allows to choose for selection criteria based on the flexibility of the machines. At this point this is the only heuristic algorithm ever introduced to solve the TFJSS ([10]).

The algorithm starts sorting jobs in non decreasing order of $r_j$. For $k = 1, ..., n$ it selects job $J_k$: if there is at least a machine able to process $J_k$ among the ones to which at least one job has already been assigned and for which the spread-time constraints would be not violated, then the algorithm chooses one of those machines via a criterion $C'$ and assigns $J_k$ to this machine; else, it chooses a new machine among the ones that are not already used via a criterion $C''$ and assigns $J_k$ to this machine.

**Algorithm 1** Heuristic [10]

Sort jobs in non decreasing order of $r_j$;

**for** $k = 1, ..., n$ **do**

**if** there is at least a machine able to process $J_k$ among the ones to which it has already been assigned at least one job and for which the spread-time constraints would be not violated, **then**

Choose one of those machines via a criterion $C'$ and assign $J_k$ to this machine;

**else** Choose a new machine among the ones that are not already used via a criterion $C''$ and assign $J_k$ to this machine.

**end if**

**end for**

It is possible to choose among three criteria for $C'$ and $C''$ before the algorithm starts:

- Minimum flexibility (cmin): choose one of the machines (to which a job has already been assigned in the case of $C'$, to which no job has already been assigned in the case of $C''$) with the minimum flexibility; i. e. able to process jobs from the minimum number of job groups and for which the spread-time constraints would be not violated;
- Maximum flexibility (cmax): choose one of the machines (to which a job has already been assigned in the case of $C'$, to which no job has already been assigned in the case of $C''$) with the maximum flexibility; i. e. able to process jobs from the maximum number of job groups and for which the spread-time constraints would be not violated;
- Random: choose one of the machines able to process that job and for which the spread-time constraints would be not violated.

### 3.4.2 Computational Experiments

The algorithm in [10] has been developed in Java language with Eclipse Jee Oxygen on a DELL Inspiron with 8 GB RAM, SSD hard disk and Windows 10 operative system. The instances for [10] have been created to be in accordance with real world situations that occur in large scale international airports. Informal talks have been carried out with accessibility and security managers in important european international airports, detecting averages and numbers of the real world cases that they face every day.

We consider a discrete time-line of $[0, 200]$, discrete randomly chosen $r_j$ in the whole time-line and integer duration of the jobs randomly chosen in $[5, 30]$. For

the experiments we put $L = 80$ or $L = 100$, $m = \frac{13}{20}n$ and $n = 100, 500, 1000$ or 2000. The jobs are divided into four groups corresponding to the four languages spoken by the machines/workers. For simplifying the notation we identify this four languages with the Italian (basic language spoken by all the machines/workers), English, Spanish and French language. The $m$ machines/workers are divided into five classes of equal cardinality containing respectively all the machines/workers speaking only the basic language of the airport (Italian language in our case), all machines/workers speaking only Italian and English, all machines/workers speaking only Italian, English and French, all machines/workers speaking only Italian, English and Spanish, all machines/workers speaking Italian, English, Spanish and French.

**Remark.** Consider an iteration of the algorithm choosing a job $J_k$. W. l. g. we suppose we are adopting criteria (cmin,cmin) for $(C', C'')$, i.e. choosing the minimum flexibility criterion cmin for both $C'$ and $C''$, and that there are no machines able to process $J_k$ among the ones to which a job has already been assigned and with minimum flexibility. Suppose that the algorithm can not find a machine (of minimum flexibility) $J_k$ until it reaches two classes of machines/workers, call it $q$ and $q'$, that can process $J_k$ and having the same cardinality. Machines from $q$ and $q'$ are able to process $J_k$ and have the same flexibility, so it does not matter to $J_k$ which one to choose, a machine/worker from $q$ or $q'$. Preferring always one of the two classes could affect the final solution. In this situation the algorithm will pick a machine/worker from $q \cup q'$. The same observation is valid also for other couples of criteria for $C', C''$. This situation corresponds to the case in which a job/passenger asking assistance with Italian or English language cannot be allocated to a machine from the two classes with minimum flexibility: the class containing machines/workers speaking Italian, English and French and the class

containing ones speaking Italian, English and Spanish have the same flexibility and can process that job. This situation can occur many times during the execution of the algorithm and the choices it makes can bring to different solutions. For this reason every instance has been solved many times.

We study 15 instances for each couple $(n, L)$ with $n = 100, 500, 1000$ and 2000 and $L = 80$ or 100. In the left part of the tables there are the numbers of jobs/passengers requiring respectively Italian (ITA), English (ENG), French (FRA) and Spanish (SPA) language. This instances represent realistic situations in which of course the number of jobs/passengers requiring the basic language of the airport (Italian in our case) or English is greater than the number of jobs/passengers requiring the other two languages. In the right part of the tables there are the results of the solution of the instances. Each row contains the averages of the results of 20 executions of the algorithm with criteria (cmin,cmin), i.e. choosing the minimum flexibility criterion cmin for both $C'$ and $C''$, for a total of 2400 executions. The reported numbers are, from left to right:

- the best value obtained for each instance (BVO)
- the average of the values obtained (AVO)
- the average of the times in ms of the initial sorting process via selection sort (SSA)
- the average of the times in ms of the main algorithm (MAA).

### 3.4.3 Conclusions

The couple of criteria (cmin,cmin) proved to be the best. All the other combinations of criteria bring to worst results and sometimes they leave some jobs/passengers unassigned, especially in the cases in which $n = 100$.

Computational study shows that the value of BVO and AVO generally depend on the distribution of the jobs during the time-line, i. e. on the number of jobs that overlap, and from the length $d_j - r_j$ of the jobs.

The results are competitive with the numbers of large scale airports, as stated by talks and comparisons with accessibility and security managers of international airports. In the real cases on which the instances with $L = 80$ are modeled the number of machines/workers currently used to process $n = 500$ jobs is generally between 150 and 160. Similarly in the case in which $n = 1000$ the number of machines/workers used to accomplish all jobs is between 290 and 310; in the case of $n = 2000$ that number is generally between 570 and 600. In the real cases on which the instances with $L = 100$ are modeled the number of machines/workers currently used to process $n = 500$ jobs is generally between 110 and 120. Similarly in the case in which $n = 1000$ the number of machines/workers used to accomplish all jobs is between 220 and 240; in the case of $n = 2000$ that number is generally between 420 and 450.

Apart from the cases in which $n = 100$ (see tables 3.1 and 3.5), for which the numbers obtained by the algorithm are very similar to the ones recorded in airports, a comparison between the real world numbers and the results of the algorithm shows that those latter seem to be better in the majority of cases (see Tables 3.2-3.4 and Tables 3.6-3.8). This stands despite the fact that in almost every airport some of the rules stated by international conventions and represented in our mathematical model are ignored, hence the numbers recorded in real world situations arise from schedulings that are not feasible for our formulation of the problem. So the results obtained by the algorithm are generally better than the ones recorded in many real world cases (see table 3.9 and 3.10 for a comparison).

Computational experiments show that at least the 40 per cent of the ma-

Table 3.1: Results for the instances with $n = 100$, $L = 80$

| Instance | ITA | ENG | FRA | SPA | BVO | AVO | SSA (ms) | MAA (ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 60 | 20 | 10 | 10 | 39 | 39,2 | 8,3 | 1 |
| 2 | 55 | 21 | 13 | 11 | 37 | 37 | 9,5 | 1 |
| 3 | 51 | 23 | 13 | 13 | 32 | 32,5 | 1,6 | 1,6 |
| 4 | 46 | 26 | 15 | 13 | 34 | 35,4 | 1,6 | 1,7 |
| 5 | 43 | 28 | 14 | 15 | 39 | 39,6 | 1,8 | 1,4 |
| 6 | 40 | 30 | 15 | 15 | 27 | 28,7 | 2,2 | 1,4 |
| 7 | 38 | 31 | 16 | 15 | 33 | 36,4 | 2 | 1,3 |
| 8 | 36 | 33 | 15 | 16 | 35 | 37,2 | 6,1 | 1,2 |
| 9 | 34 | 33 | 17 | 16 | 38 | 39,1 | 2,5 | 1,5 |
| 10 | 33 | 33 | 17 | 17 | 34 | 35 | 2,9 | 1,7 |
| 11 | 31 | 34 | 16 | 19 | 31 | 33,5 | 8,7 | 1 |
| 12 | 32 | 30 | 19 | 19 | 39 | 39 | 10,1 | 1 |
| 13 | 31 | 30 | 20 | 19 | 33 | 33 | 9,1 | 1 |
| 14 | 30 | 29 | 21 | 20 | 35 | 36,4 | 8,5 | 1 |
| 15 | 27 | 28 | 21 | 24 | 31 | 33,2 | 9 | 1 |

chines/workers is saved; moreover in the majority of cases the number of machines/workers saved reaches notable levels with more than the 60 per cent of machines/workers saved in the case $L = 80$ (see Table 3.4, instance 1) and 71 per cent in the case $L = 100$ (see Table 3.8, instance 6).

Table 3.2: Results for the instances with $n = 500$, $L = 80$

| Instance | ITA | ENG | FRA | SPA | BVO | AVO | SSA (ms) | MAA (ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 300 | 100 | 50 | 50 | 134 | 138,6 | 26,1 | 14,6 |
| 2 | 279 | 109 | 52 | 60 | 146 | 148,8 | 22,7 | 14,3 |
| 3 | 252 | 133 | 60 | 55 | 135 | 136,7 | 25,7 | 13,4 |
| 4 | 231 | 142 | 66 | 61 | 152 | 153 | 24,2 | 11,3 |
| 5 | 212 | 146 | 72 | 70 | 134 | 136,1 | 23,1 | 18,3 |
| 6 | 200 | 150 | 75 | 75 | 139 | 142,3 | 26,5 | 13,7 |
| 7 | 189 | 150 | 80 | 81 | 159 | 160,7 | 26,2 | 17,4 |
| 8 | 169 | 141 | 100 | 90 | 148 | 149,1 | 23,1 | 13,7 |
| 9 | 153 | 155 | 91 | 101 | 130 | 132,7 | 23 | 16,1 |
| 10 | 158 | 140 | 103 | 99 | 152 | 154,6 | 24,5 | 13,6 |
| 11 | 147 | 134 | 111 | 108 | 158 | 158,7 | 23,2 | 14,5 |
| 12 | 140 | 142 | 108 | 110 | 154 | 156,6 | 24,8 | 15,2 |
| 13 | 141 | 139 | 110 | 110 | 160 | 160,9 | 24,8 | 15,2 |
| 14 | 138 | 133 | 114 | 115 | 133 | 138,3 | 23 | 14,2 |
| 15 | 130 | 134 | 120 | 116 | 125 | 128 | 23,8 | 14,3 |

Table 3.3: Results for the instances with $n = 1000$, $L = 80$

| Instance | ITA | ENG | FRA | SPA | BVO | AVO | SSA (ms) | MAA (ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 600 | 200 | 100 | 100 | 273 | 275,2 | 43,8 | 21,1 |
| 2 | 570 | 212 | 110 | 108 | 276 | 278,1 | 48,4 | 18.3 |
| 3 | 521 | 241 | 117 | 121 | 304 | 304,7 | 46,6 | 20,5 |
| 4 | 462 | 262 | 140 | 136 | 292 | 295,2 | 45,1 | 19,9 |
| 5 | 421 | 289 | 150 | 140 | 287 | 289,4 | 46,9 | 20,9 |
| 6 | 400 | 300 | 150 | 150 | 292 | 294,7 | 41,3 | 21,9 |
| 7 | 371 | 330 | 152 | 147 | 273 | 276,2 | 52 | 16,7 |
| 8 | 363 | 331 | 150 | 156 | 277 | 282,1 | 46,5 | 14,1 |
| 9 | 351 | 317 | 165 | 167 | 264 | 266,2 | 43,3 | 14,3 |
| 10 | 324 | 342 | 171 | 163 | 277 | 279,9 | 43,6 | 14,7 |
| 11 | 330 | 302 | 190 | 178 | 285 | 286,3 | 45,5 | 14,3 |
| 12 | 310 | 311 | 179 | 200 | 289 | 291,7 | 42,9 | 14,2 |
| 13 | 303 | 298 | 204 | 195 | 288 | 289,8 | 43 | 13,1 |
| 14 | 285 | 287 | 200 | 228 | 280 | 284,3 | 40,7 | 13 |
| 15 | 272 | 258 | 240 | 230 | 278 | 281,2 | 41,1 | 12,3 |

Table 3.4: Results for the instances with $n = 2000$, $L = 80$

| Instance | ITA | ENG | FRA | SPA | BVO | AVO | SSA (ms) | MAA (ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1200 | 400 | 200 | 200 | 522 | 523,9 | 60,6 | 38,5 |
| 2 | 1114 | 429 | 223 | 234 | 562 | 565,3 | 59,5 | 36,1 |
| 3 | 1058 | 457 | 245 | 240 | 552 | 555,3 | 61 | 36,6 |
| 4 | 950 | 536 | 258 | 256 | 557 | 558,4 | 57,3 | 37,1 |
| 5 | 859 | 571 | 291 | 279 | 526 | 530,1 | 64,1 | 39,1 |
| 6 | 800 | 600 | 300 | 300 | 560 | 562,3 | 66,1 | 36,5 |
| 7 | 772 | 604 | 319 | 305 | 592 | 593,6 | 63 | 38,2 |
| 8 | 729 | 580 | 345 | 346 | 533 | 535,9 | 69,3 | 41,6 |
| 9 | 708 | 572 | 358 | 362 | 529 | 532,4 | 62,6 | 41,1 |
| 10 | 680 | 553 | 385 | 382 | 561 | 563,7 | 63,6 | 37,7 |
| 11 | 651 | 546 | 405 | 398 | 531 | 534,4 | 61,8 | 38,7 |
| 12 | 619 | 539 | 421 | 421 | 553 | 556 | 60,3 | 31,6 |
| 13 | 605 | 520 | 439 | 436 | 539 | 541,4 | 59,4 | 30,3 |
| 14 | 582 | 523 | 450 | 445 | 558 | 559,5 | 59,5 | 33,1 |
| 15 | 558 | 511 | 462 | 469 | 567 | 569,3 | 66,4 | 37,3 |

Table 3.5: Results for the instances with $n = 100$, $L = 100$

| Instance | ITA | ENG | FRA | SPA | BVO | AVO | SSA (ms) | MAA (ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 60 | 20 | 10 | 10 | 28 | 28,8 | 2,3 | 1,8 |
| 2 | 55 | 22 | 11 | 12 | 31 | 31 | 8,1 | 1 |
| 3 | 50 | 24 | 13 | 13 | 27 | 27,1 | 3,9 | 1,3 |
| 4 | 46 | 25 | 15 | 14 | 29 | 29,2 | 2,1 | 1,8 |
| 5 | 42 | 29 | 14 | 15 | 26 | 26,9 | 2,6 | 1,8 |
| 6 | 40 | 30 | 15 | 15 | 31 | 31 | 2,4 | 1,7 |
| 7 | 38 | 31 | 16 | 15 | 23 | 23,9 | 2,1 | 1,5 |
| 8 | 36 | 30 | 18 | 16 | 28 | 29 | 3 | 1,6 |
| 9 | 34 | 33 | 17 | 16 | 30 | 30,2 | 2,7 | 1,6 |
| 10 | 33 | 32 | 18 | 17 | 29 | 29 | 2,6 | 1,5 |
| 11 | 33 | 30 | 20 | 17 | 25 | 26,7 | 2,5 | 1,6 |
| 12 | 31 | 31 | 18 | 20 | 29 | 29 | 9,5 | 1 |
| 13 | 31 | 30 | 21 | 18 | 32 | 33,6 | 9,8 | 1 |
| 14 | 29 | 28 | 23 | 20 | 24 | 24,8 | 10,3 | 1 |
| 15 | 28 | 27 | 24 | 21 | 27 | 28,3 | 9,1 | 1 |

Table 3.6: Results for the instances with $n = 500$, $L = 100$

| Instance | ITA | ENG | FRA | SPA | BVO | AVO | SSA (ms) | MAA (ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 300 | 100 | 50 | 50 | 110 | 111,7 | 20,6 | 14,7 |
| 2 | 273 | 111 | 59 | 57 | 107 | 108,7 | 21,7 | 15,6 |
| 3 | 249 | 120 | 67 | 64 | 112 | 114,4 | 21,8 | 12,7 |
| 4 | 230 | 129 | 73 | 68 | 105 | 106,6 | 24,9 | 14,4 |
| 5 | 212 | 140 | 73 | 75 | 107 | 109,8 | 22,4 | 14 |
| 6 | 200 | 150 | 75 | 75 | 102 | 104,9 | 25,2 | 14,2 |
| 7 | 186 | 149 | 83 | 82 | 104 | 105,9 | 21,4 | 12,7 |
| 8 | 165 | 144 | 99 | 92 | 99 | 101,9 | 23,7 | 12,9 |
| 9 | 153 | 156 | 90 | 101 | 107 | 108,7 | 22 | 15,3 |
| 10 | 156 | 140 | 99 | 105 | 98 | 100 | 24 | 13,7 |
| 11 | 147 | 133 | 113 | 107 | 103 | 105 | 22,3 | 14,8 |
| 12 | 143 | 130 | 116 | 111 | 101 | 102,4 | 22,6 | 11,7 |
| 13 | 136 | 140 | 111 | 113 | 108 | 109,1 | 20,5 | 12,9 |
| 14 | 135 | 132 | 124 | 109 | 97 | 98,9 | 21,9 | 13,3 |
| 15 | 133 | 124 | 127 | 116 | 98 | 100,3 | 23,3 | 13,7 |

Table 3.7: Results for the instances with $n = 1000$, $L = 100$

| Instance | ITA | ENG | FRA | SPA | BVO | AVO | SSA (ms) | MAA (ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 600 | 200 | 100 | 100 | 192 | 194 | 44,1 | 12 |
| 2 | 563 | 217 | 117 | 103 | 199 | 201,6 | 44,8 | 16,5 |
| 3 | 520 | 242 | 124 | 114 | 210 | 210,5 | 45 | 14,5 |
| 4 | 459 | 261 | 144 | 136 | 194 | 197,6 | 43,5 | 14,2 |
| 5 | 425 | 280 | 145 | 150 | 200 | 201,9 | 45 | 17,4 |
| 6 | 400 | 300 | 150 | 150 | 209 | 211,4 | 41,4 | 16,5 |
| 7 | 382 | 307 | 160 | 151 | 206 | 209,3 | 43,1 | 18,8 |
| 8 | 365 | 299 | 170 | 166 | 199 | 199,7 | 40,8 | 17 |
| 9 | 348 | 298 | 180 | 174 | 220 | 222,1 | 41,7 | 18,1 |
| 10 | 325 | 345 | 161 | 169 | 200 | 200,75 | 41,8 | 16,1 |
| 11 | 320 | 290 | 198 | 192 | 190 | 193,2 | 42,8 | 15,9 |
| 12 | 311 | 289 | 194 | 206 | 201 | 203,2 | 44,2 | 15,9 |
| 13 | 302 | 281 | 206 | 211 | 195 | 197,4 | 42,3 | 15,2 |
| 14 | 285 | 275 | 221 | 219 | 202 | 205,1 | 47,6 | 13,6 |
| 15 | 279 | 253 | 257 | 211 | 195 | 197,1 | 44,1 | 11,1 |

Table 3.8: Results for the instances with $n = 2000$, $L = 100$.

| Instance | ITA | ENG | FRA | SPA | BVO | AVO | SSA (ms) | MAA (ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1200 | 400 | 200 | 200 | 382 | 385,8 | 62,6 | 30,8 |
| 2 | 1103 | 438 | 228 | 231 | 385 | 387,8 | 59,5 | 28,4 |
| 3 | 1026 | 481 | 253 | 240 | 371 | 376,3 | 58,6 | 28,4 |
| 4 | 948 | 539 | 262 | 251 | 387 | 393,6 | 56,9 | 28,8 |
| 5 | 852 | 569 | 288 | 291 | 386 | 388,5 | 59,4 | 32,9 |
| 6 | 800 | 600 | 300 | 300 | 370 | 374,1 | 63,2 | 31,1 |
| 7 | 783 | 598 | 302 | 317 | 376 | 379,5 | 61,6 | 28,8 |
| 8 | 756 | 601 | 327 | 316 | 395 | 399,3 | 62,5 | 31,1 |
| 9 | 730 | 582 | 348 | 340 | 380 | 386,9 | 61 | 34,4 |
| 10 | 703 | 580 | 361 | 356 | 379 | 381,9 | 65,1 | 36,9 |
| 11 | 660 | 681 | 335 | 324 | 382 | 385,8 | 65,1 | 37,8 |
| 12 | 641 | 555 | 394 | 410 | 386 | 388,1 | 59,2 | 25,6 |
| 13 | 606 | 530 | 447 | 417 | 388 | 391,5 | 62,1 | 29,2 |
| 14 | 586 | 521 | 450 | 443 | 396 | 396,7 | 67 | 34,2 |
| 15 | 553 | 509 | 481 | 457 | 380 | 383,1 | 61 | 28 |

Table 3.9: Comparison in the case $L = 80$

| $L = 80$ | Case $n = 500$ | Case $n = 1000$ | Case $n = 2000$ |
|---|---|---|---|
| Algorithm | 125 - 160 | 264 - 304 | 522 - 592 |
| Real case | 150 - 160 | 290 - 310 | 570 - 600 |

Table 3.10: Comparison in the case $L = 100$

| $L = 100$ | Case $n = 500$ | Case $n = 1000$ | Case $n = 2000$ |
|---|---|---|---|
| Algorithm | 97 - 112 | 190 - 220 | 370 - 396 |
| Real case | 110 - 120 | 220 - 240 | 420 - 450 |

**Acknowledgments**

I would like to thank my dear friends and colleagues Anna Tomeo BS and Francesco Garofalo BS for their fundamental support in the darkest moments of this adventure.

I am also grateful to Eng. Ersilia Vallefuoco for her constant encouragement throughout my PhD experience.

I would also like to thank the SInAPSi Centre (Centre for the Active and Participatory Inclusion of Students) of the University of Naples "Federico II", a true international excellence in helping students with special needs, for their remarkable contribution to my PhD program: they helped me to find ways to overcome my sight problems by the use of accessible technologies in order to work and to perform at my best.

# Bibliography

[1] Ford, L.R., Fulkerson, D.R.: Flows in networks, Princeton University Press, Princeton, New Jersey (1962)

[2] Gertsbakh, I., Stern, H.I., Minimal resources for fixed and variable job schedules, Operations Research **26**(1), 68–85 (1978)

[3] Gupta, U.I., Lee, D.T., Leung, J.Y.-T.: An optimal solution for the channel-assignment problem. In: IEEE Transactions on Computers C, pp. 807–810. IEEE (1979)

[4] Fischetti, M., Martello, S., Toth, P.: The Fixed Job Schedule problem with Spread-Time Constraints, Operations Research **35**(6), 849–858 (1987)

[5] Fischetti, M., Martello, S., Toth, P.: The Fixed Job Schedule problem with Working-Time Constraints, Operations Research **37**(3), 395–403 (1989)

[6] Fischetti, M., Martello, S., Toth, P.: Approximation Algorithms for Fixed Job Schedule Problems, Operations Research **40**(1-supplement-1), S96–S108 (1992)

[7] Kolen, A., Kroon, L.: License class design:complexity and algorithms, European Journal of Operational Research **63**(3), 432–444 (1992)

[8] Kroon, L., Salomon, M., Van Wassenhove, L.: Exact and approximation algorithms for the tactical fixed interval scheduling problem, Operations Research **45**(4), 624–638 (1997)

[9] Zhou, S., Zhang, X., Chen, B., van de Velde, S.: Tactical Fixed Job Scheduling with Spread-Time Constraints, Computers & Operations Research **47**, 53–60 (2014)

[10] Mele, M., Festa, P.: Scheduling assistance for passengers with special needs in large scale airports. In: 9th International Conference on Computational Logistics, pp. 388–400 (2018)

[11] Reinhardt, L.B., Clausen, T., Pisinger, D.: Route planning for airport personnel transporting with reduced mobility, DTU Management Engineering, pp.1–21 (2010)

[12] Kovalyov, M.Y., Ng., C.T., Chen, T.C.E.: Fixed interval scheduling: Models, applications, computational complexity and algorithms, European Journal of Operational Research **178**(2), 331–342 (1992)

[13] Kolen, A., Lenstra, J., Papadimitriou, C., Spieksma, F.: Interval scheduling: A survey, Naval Research Logistics (NRL) **54**(5), 530–543 (1992)

[14] Keil, M.: On the complexity of scheduling tasks with discrete starting times, Operations research letters **12**(5), 293–295 (1992)

[15] Huang, Q., Lloyd, E.: Cost Constrained Fixed Job Scheduling. In: 8th Italian Conference on Theoretical Computer Science, pp. 111–124 (2013)

[16] Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms, Cambridge University Press, New York, U.S.A. (2011)

[17] Scholl, A.: Balancing and Sequencing of Assembly Lines, Springer-Verlag, Heidelberg (1999)

[18] Martello, S., Toth, P.: A heuristic approach to the bus driver scheduling problem, European Journal of Operational Research **24**(1), 106–117 (1986)

[19] De Leone, R., Festa, P., Marchitto, E.: A Bus Driver Scheduling Problem: a new mathematical model and a GRASP approximate solution, Springer Science+Business Media **17**(4), 441–466 (2011)

[20] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness , W.H. Freeman and Company, New York, U.S.A. (1979)

[21] Cabrera, G.G., Rubio, J. M. L.: Hybrid Algorithm of Tabu Search and Integer Programming for the Railway Crew Scheduling Problem. In: 2th Asia-Pacific Conference on Computational Intelligence and Industrial Applications, pp. 413–416 (2009)

[22] Wren, A., Rousseau, J. M.: Bus Driver Scheduling - An Overview, School of Computer Studies Research Report Series, University of Leeds **93**(31), 1–14 (1993)

[23] Portugal, R., Lourenco, H. R., Paixao, J. P.: Driver scheduling problem modelling, Public Transport, Springer-Verlag **1**(2), 103–120 (2008)

[24] Rodrigues, M. M., de Souza, C. C., Moura, A. V.: Vehicle and crew scheduling for urban bus lines, European Journal of Operational Research **170**(3), 844–862 (2006)

[25] Mastelic, T., Fdhila, W., Brandic, I., Rinderle-Ma, S.: Predicting Resource Allocation and Costs for Business Processes in the Cloud. In: 11th World Congress on Services, pp. 47–54 (2015)