UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

Dottorato in Matematica e Informatica (XXXVI ciclo)

Settore Disciplinare: MAT/09 - Ricerca Operativa

Tesi di Dottorato

Balancing the average weighted completion times of two classes of jobs: a new scheduling problem

Matteo Avolio

Supervisore
Prof. Antonio Fuduli

Coordinatore
Prof. Giorgio Terracina

A.A. 2022/2023

Abstract

Exploring a new area of the scheduling theory and inspired by a real application in an academic context, in this thesis we introduce a new single-machine two-agent scheduling problem, aimed at balancing the average weighted completion times of two different classes of jobs, one per agent. Differently from the common multiagent cases, which are generally of the competing type, this problem could be interpreted as a cooperative type problem. In fact, even if the two agents share the same machine, they cooperate to optimize the unique global objective function, in order to balance their average weighted completion times.

While for the case with identical jobs and unitary weight we present an exact algorithm providing an optimal solution in linear time, for the general case we prove the NP-hardness of the problem and we propose a mathematical formulation as a variant of the well known quadratic assignment problem. By applying the Glover linearization, we obtain a mixed integer linear program exploited to design a Lagrangian heuristics based on solving, at each iteration, a linear assignment problem. Since the proposed algorithm has revealed to be able to solve instances up to 500 jobs, in order to face larger scale instances (up to 2000 jobs) we also propose a genetic algorithm.

Introduction

Every day many manufacturing and service companies have to deal with the allocation of resources to tasks over given time periods. This kind of decision making process, aimed at optimizing one or more objectives, is named *scheduling*.

The aim of scheduling theory is to study these problems from a mathematical point of view, providing optimization models and resolution methods to automatize this decision making process. Then, given their importance, scheduling problems have been deeply studied in the literature and, nowadays, scheduling theory is a well established optimization field. Scientists have obtained many important results that, each day, help industries in the achievement of their goals. Given the huge number of possible applications, the field is exponentially growing, in connection also with the evolution of the modern society, characterized by new and ever more complex problems to be efficiently solved.

The research in this field usually consists not only in designing new and more efficient algorithms aimed at solving existing problems, but also in the definition of additional problems (generally raising in the practical contexts) to be solved by means of exact or heuristic algorithms. In fact, an interesting aspect of scheduling theory resides also in the computational complexity of this kind of problems: many of them are proved to be NP-hard and, in such cases, a particular effort is often done to design efficient heuristic algorithms.

Exploring a new area of the scheduling theory and inspired by a real application in an academic context, in this thesis we introduce a new single-machine scheduling problem characterized by two decision makers. Since in the scheduling literature the decision makers are commonly named *agents*, this problem falls in the class of the so called *multiagent* scheduling problems, whose main difficulty resides in the fact that the agents have to share the same machines.

In particular, the objective of our new problem, called in the sequel BAWCT-

2 (Balancing the Average Weighted Completion Times of 2 agents) is to balance the average weighted completion times of two different classes of jobs, one per agent. Differently from the common multiagent scheduling problems, which are generally of the competing type (since the job set of each agent contributes only to the corresponding agent's objective function), BAWCT-2 could be interpreted instead as a cooperative type problem. In fact, even if the two agents share the same machine, they cooperate to optimize the (unique) global objective function, aimed at balancing their average weighted completion times.

The thesis, which is mainly constituted by the first three parts, is organized as follows.

In the first part, Chapters 1-3, we provide the reader with an introduction to scheduling theory. In particular, in the first chapter we discuss its importance and its impact on the industrial sector, by providing some examples of real life applications. In the second chapter, we present some well established results in the literature related to some single-machine scheduling problems aimed at minimizing the total completion time or the maximum lateness. We conclude this part with Chapter 3, in which we introduce the multiagent scheduling problems, discussing also the possibility of cooperation among the agents.

In the second part of the thesis, Chapters 4-9, we discuss the novelty of our work by introducing the new single-machine scheduling problem for balancing the average weighted completion times of two classes of jobs. In particular, in Chapter 4 we define the problem (providing also some possible applications) and we prove its NP-hardness by means of a polynomial reduction from the partition problem. Then, in Chapter 5, we present a quadratic assignment mathematical formulation of the problem, that we reduce to a mixed integer linear program by means of the Glover linearization technique. In Chapter 6, we focus on a simplified version of the problem, assuming identical jobs (i.e. with the same processing time) and unitary weight: for such a problem, we propose an exact solution algorithm, showing that the problem is solvable in linear time. In Chapter 7, exploiting the mixed integer linear formulation reported in Chapter 5, we design a Lagrangian heuristics based on solving, at each iteration, a linear assignment problem. In order to face large scale instances, in Chapter 8 we propose a genetic algorithm to speed up the resolution process. We conclude the second part of the thesis by Chapter 9, in which we describe some numerical experiments performed to evaluate the performance

of the Lagrangian relaxation approach and of the genetic algorithm.

In the third part of the thesis, Chapter 10, we draw some conclusions and we discuss some possible future research.

The entire work is completed by the last two parts, aimed at providing the reader with some bibliographic references (Part IV) and with two publications (Part V) drawn from the research part of the thesis. In particular:

- The contents of Chapter 6 have been published in:
  - M. Avolio, A. Fuduli. "A subset-sum type formulation of a two-agent single-machine scheduling problem". Information Processing Letters, 155, article 105886, 2020.
- The contents of Chapters 5 and 7 have been published in:
  - M. Avolio, A. Fuduli. "A Lagrangian heuristics for balancing the average weighted completion times of two classes of jobs in a single-machine scheduling problem". EURO Journal on Computational Optimization, 10, article 100032, 2022.
- The contents of Chapter 8 will be object of a possible new paper, whose writing is currently in progress.

Notation

$\stackrel{\triangle}{=}$	by definition
J	overall job set
M	overall machine set
n	number of jobs in J
m	number of machines in M
q	number of agents
π	job schedule (sequence)
$\langle j \rangle$	position of job j
$[j]$	job processed in position j
J_A	job set of agent A
J_B	job set of agent B
n_A	number of jobs in J_A
n_B	number of jobs in J_B
$\lfloor r \rfloor$	the nearest integer number less than or equal to r
$\mathbb{R}^n$	Euclidean n-dimensional space
$\partial f(x)$	subdifferential of function f at x
$\|x\|$	Euclidean norm of x

Index

Part I

Introduction to Scheduling Theory

Chapter 1

The Scheduling Problems

In this chapter, we introduce the reader to the world of scheduling theory, by discussing the importance of this decision-making process and its impact on the industrial sector. We also provide some examples of real life scheduling problems, concluding the chapter with the description of the main notation generally adopted in this field.

1.1 The Role of Scheduling

Every day many manufacturing and service companies have to deal with the allocation of resources to tasks over given time periods. This kind of decision making process, aimed at optimizing one or more objectives, is named *scheduling* [42].

Resources, tasks and objectives can assume different forms, depending on the context. For example, possible resources are machines in an industry, runways at an airport, computers in a laboratory, or professors in a university. Corresponding tasks may be operations in a production environment, take-offs and landings at the airports, processes in a computer, or courses in an academic context.

An important role in scheduling theory is played by the objectives, which define the criteria used by the decision maker to evaluate the performance of the schedule. On the basis of the practical contexts, one can adopt different objective functions, such as the minimization of the time when the last task is completed, or the minimization of the number of tasks completed after a certain time.

Then, it is easy to guess that the difficulty of solving a scheduling problem is generally related to the specific characteristics of the following three items: the resources, the tasks and the objectives. For example, there are scheduling problems characterized by just one or various resources, which in turn can be homogenous or heterogeneous. On the other hand, each task can be more or less complex in function of its features (priority level, due date, release time, and so on). There exist also scheduling problems with more than one single objective, or even multiagent scheduling problems where different sets of tasks (one per agent) have their own evaluation criteria, often competing each with the other.

Scheduling theory is aimed at studying these problems from a mathematical point of view, providing optimization models and resolution methods to automatize this decision making process. Then, given their importance, scheduling problems have been deeply studied in the literature and, nowadays, scheduling theory is a well established optimization field. Scientists have obtained many important results that, each day, help industries in the achievement of their goals. Given the huge number of possible applications, the field is exponentially growing, in connection also with the evolution of the modern society, characterized by new and ever more complex problems to be efficiently solved.

The research in this field usually consists not only in designing new and more efficient algorithms aimed at solving existing problems, but also in the definition of additional problems (generally raising in the practical contexts) to be solved by means of exact or heuristic algorithms. In fact, an interesting aspect of scheduling theory resides also in the computational complexity of this kind of problems: many of them are proved to be NP-hard and, in such cases, a particular effort is often done to design efficient heuristic algorithms.

In the next section, we will present in detail some practical examples of scheduling problems. Even if the overall notation adopted throughout the thesis is detailed in Section 1.3, we anticipate here that, using the scheduling terminology, from now on each resource will be called *machine* and each task will be named *job* (we say that a machine *processes* a job). Each job is always characterized by at least its *processing time* and its *completion time*: while the former is the time needed by the job to be processed on a given machine, the latter is a sequence dependent quantity corresponding to the time when the entire processing of the job is completed.

1.2 Some Examples

Example 1.2.1. (Toy example: the fish fry problem) A restaurant has to prepare a fish fry made of three kinds of fish: prawns, squids and anchovies. Assuming that the restaurant has only one pan and that each fish has to be cooked separately, the objective is to determine in which order to prepare them with the aim of minimizing the elapsed time between the time in which the first kind of fish is ready and the time in which the whole fish fry can be served (i.e. all the fishes have been cooked). In this toy problem, the pan is the machine (in this case just one), while the three types of fish represent the jobs, with the processing time being their corresponding cooking time.

This problem can be easily solved by noting that the time in which the fish fry will be ready is known, since it is easily computable by summing the time needed to cook all the kinds of fish. Recalling that the objective is to shrink the interval between this time and the time in which the first kind of fish is ready, the optimal solution can be obtained by cooking first the fish characterized by the highest cooking time. The order in which the other two types of fish will be cooked is completely irrelevant. ||

Note that the toy problem described in Example 1.2.1 is solvable also by inspection, i.e. by evaluating all the six (i.e. 3!) possible schedules and by selecting the most convenient one. We observe in fact that the scheduling problems are intrinsically of combinatorial nature: this means that passing, for example, from three to ten types of fish makes the problem intractable to be solved by inspection since we have to evaluate 10! possible different schedules.

Example 1.2.2. (Gate assignments at an airport [42]) Each major airport has a various number of terminals, each of them consisting of dozens of gates handling, daily, hundreds of planes (arriving and departing). The gates, as the planes, are not identical: there are different kinds of gates depending on the available space and different kinds of planes depending on the size. This implies that a plane has to be accommodated in a gate that can actually host it. When a plane arrives at a gate, some operations have to be performed:

the arriving passengers have to be deplaned, the plane has to be serviced and the new passengers have to be boarded. Planes arrive and depart following a certain schedule; the departure time, in this case, can be interpreted as a due date and the airline performance is measured accordingly. In this scenario the gates are the machines and the planes are the jobs, whose starting and completion times coincide, respectively, with the arrivals of the planes at the gates and with their departures. The scheduling problem consists in assigning the planes to the gates (that are both suitable and available at the respective arrival times), optimizing at the same time some objectives, such as the minimization of eventual delays of the planes. ||

Example 1.2.2 describes a simplified version of a more general problem. In fact, in real cases it is easy to imagine that an important role is played by some kind of randomness, such as the weather that could cause problems in handling the passengers or other unforeseen events at the airports.

Example 1.2.3. (*Scheduling programs in a CPU [42]*) One of the most important components inside a multi-tasking computer operating system is the scheduler. As its name suggests, it has to schedule the programs (the jobs) to be executed, deciding the time devoted to each of them by the CPU (the machine). What makes this scenario really interesting is that the exact processing times (i.e. the duration of the programs) are not known in advance: their distribution, together with their means and variances, is usually given. In this case, a possible objective function is given by the minimization of the expected weighted sum of the completion times of all the tasks, where the weight of each task is given by a priority value specified by the user.

Since a program can last too much, to avoid congestion in the system, usually the operating system rotates the tasks, processing each time only small parts of them. An interruption of the processing of a task, crucial in this scenario but used in many scheduling contexts, is referred to as *preemption* as we will see in the next section. ||

1.3 Glossary and Notation

In this section, we present the basic scheduling notation adopted throughout the thesis, referring to the *system* as the environment where the machines are available to process the jobs from time $t = 0$.

In particular, we will always assume that both the number of jobs and the number of machines are finite and that, at any time, each machine is able to process at most one job. We will denote by $J \triangleq \{1, \ldots, n\}$ the set of jobs and by $M \triangleq \{1, \ldots, m\}$ the overall set of the available machines; additionally, unless differently specified, the index i will be referred to the ith machine, while the index j to the jth job.

The most common quantities characterizing a generic job j are the following:

- **Processing time** (p_{ij}): it represents the time required by machine i for processing job j. The index i is omitted for single-machine scheduling problems or in case the processing time of job j is machine independent.
- **Release date** (r_j): it indicates the time when job j enters the system, being ready to be processed.
- **Due date** (d_j): it denotes the time within the processing of job j has to be possibly concluded for not resulting in a penalty. It is a sort of soft version of the *deadline*, that forces the system to conclude the job processing within a certain peremptory time.
- **Weight** (w_j): it measures the importance of job j with respect to the other jobs of the system.
- **Machine set** (M_j): it denotes the set of machines that can process job j.

For describing a particular scheduling problem, we will use the triple (α, β, γ) by adopting the following Graham notation (see [25]), very used in the literature:

$$\alpha|\beta|\gamma.$$

The first field α regards the machine environment and defines for example number and type of the available machines. The second field β describes the job environment, specifying all the quantities and/or relationships and/or

constraints characterizing the jobs: such field is empty in the simplest case, where the jobs are characterized by only the processing time. Finally, the last field γ lists the objective functions to be optimized.

In the following, we will mention some possible characteristics we can find in each of the three above fields. In α, for example, we can have:

- **Single-machine** (1): it represents the simplest situation, in which the system is equipped with a unique machine.

- **Parallel identical machines** (Pm): there are m identical machines working in parallel and any job j requires to be processed indifferently by one of the m machines.

  A simple example of identical machines working in parallel is constituted by the cashiers in a supermarket: in this case $M_j = M$, for each $j = 1, \ldots, n$, since, in general, each client (corresponding to a job) can be processed by any cashier.

- **Flow shop** (Fm): the system is characterized by the presence of m machines in series and all the jobs must be processed by the m machines, following the same order in passing from a machine to another one.

  A flow shop problem with $m = 2$ is in a day hospital ward, when all the patients (the jobs) have to first undergo a specialist visit (first machine), followed by a computerized axial tomography (second machine).

- **Job shop** (Jm): each job must be processed by all the m machines following a specific order of its own. It differs from the flow shop framework because, in this case, each job can follow a different order on the machines with respect to the other jobs.

- **Open shop** (Om): in this case, each job has to be processed by all the m machines without any specific order.

  An example of a two-machine open shop problem is given by a set of students (the jobs) who have to take an oral exam constituted by two independent parts with two respective different teachers. In this case, for any student, it is irrelevant in which order the two oral parts are carried out.

Possible quantities or constraints populating the field β are:

- **Release date** (r_j): in a scheduling problem with release dates, a job j cannot be processed before its corresponding release date r_j. A scheduling problem in which the field β contains the release dates is said to be *dynamic*, since it is assumed that the jobs can enter the system in different moments, otherwise it is called *static* (i.e. all the jobs are available at time $t = 0$).

- **Preemption** (prmp): it allows the possibility of interrupting the processing of a job on a certain machine. Such processing must be completed either later on the same machine, or (immediately or later) on another identical machine.

- **Precedence constraints** (prec): such constraints impose that the processing of a job cannot start before completing the processing of eventual other jobs. In particular, when job j must precede job k, we write $j \rightarrow k$.

- **Setup time** (s_{ijk}): it represents the time needed to set up machine i, before processing job k immediately after job j. In particular, s_{i0k} is the setup time of machine i when processing job k as the first job, while s_{ij0} is the clean up time, i.e. the time needed for cleaning machine i after processing job j as the last job. In case the setup time among two jobs, j and k, is independent of the machine, we use the notation s_{jk}.

Concerning the objective functions to be optimized (field γ), they are always functions of the completion times, which in turn are sequence dependent quantities. The *completion time* of job j on machine i, denoted by C_{ij}, is the time when the processing of job j on machine i is completed. On the other hand, C_j indicates the *generic completion time* of job j, which corresponds to the time when job j leaves the system.

The following quantities, depending on the completion times and on the due dates, are useful to define possible examples of objective functions:

- The *lateness* L_j of job j is defined as $L_j \stackrel{\triangle}{=} C_j - d_j$. Note that this value is strictly positive when job j is completed late, strictly negative when job j is completed early and zero if it is punctually completed.

- The *tardiness* T_j of job j is defined as $T_j \stackrel{\triangle}{=} \max\{0, L_j\}$. Note that the *tardiness* is always nonnegative and it is strictly positive if job j is completed late.

- The *unit penalty* U_j of job j is defined as

$$U_j \triangleq \begin{cases} 1 & \text{if } L_j > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then, recalling that n is the number of jobs, possible objective functions are the following:

- **Makespan** (C_{max}): it is the maximum completion time, i.e.

$$C_{max} \triangleq \max\{C_1, \ldots, C_n\}.$$

  In the single-machine case, it coincides with the time in which the job processed in the last position leaves the system, while, in the case of more machines, it coincides with the time in which the processing of all the jobs terminates.

- **Maximum Lateness** (L_{max}): it measures the maximum due date violation, i.e.

$$L_{max} \triangleq \max\{L_1, \ldots, L_n\}.$$

- **Maximum Tardiness** (T_{max}): it measures the maximum tardiness among the jobs, i.e.

$$T_{max} \triangleq \max\{T_1, \ldots, T_n\}.$$

- **Total completion time** ($\sum_{j=1}^{n} C_j$): it gives an indication of the *inventory* cost of the jobs, because, assuming the system to be static, it coincides with the sum of the time spent in the system by all the jobs. In the literature, the sum of the completion times is often referred to as *flow time*.

- **Total weighted completion time** ($\sum_{j=1}^{n} w_j C_j$): it is a generalization of the previous item, obtained by multiplying the completion time of each job by the corresponding weight. In the literature, it is commonly named *weighted flow time*.

- **Total weighted tardiness** ($\sum_{j=1}^{n} w_j T_j$): it takes into account the tardiness of each job, resulting in a more general objective function with respect to the previous one.

- **Weighted number of tardy jobs** ($\sum_{j=1}^{n} w_j U_j$): it is a particular measure, representing the percentage of the on time items, often used for evaluating the manager's performance.

Note that all the above objectives are referred to as *regular performance* measures, since they are nondecreasing functions with respect to the completion times $C_1, \ldots, C_n$.

We conclude the chapter by remarking that, unless differently specified, a generic schedule of the jobs will be denoted by π and the job processed in the jth position of π by $[j]$, i.e.

$$\pi = ([1], \ldots, [n]).$$

Consequently, by $\cdot_{[j]}$ we will indicate any quantity related to the job processed in position j. Finally, in the case of nonpreemptive single-machine problems, a schedule π is also referred to as a sequence, i.e. a permutation of the n jobs.

Chapter 2

Some Deterministic Single-Machine Problems

In this chapter, we present some deterministic single-machine scheduling problems, characterized by the minimization of the total completion time or of the maximum lateness.

These problems, which are well established in the literature, are useful for the reader to enter the world of the scheduling theory. For better readability, each result is also provided with an example and a graphical representation.

2.1 The Total Completion Time

Consider the following scheduling problem:

$$1||\sum_{j=1}^{n} C_j. \tag{2.1}$$

Theorem 2.1.1. (Shortest Processing Time (SPT) rule [48]) *The optimal solution to problem (2.1) is obtained by scheduling the jobs in a non-decreasing order with respect to their processing time, i.e.*

$$p_{[1]} \leq p_{[2]} \leq \ldots \leq p_{[n]}. \tag{2.2}$$

Proof. From the definition of C_j, for each job $j = 1, \ldots, n$, we know that:

$$C_{[1]} = p_{[1]}$$

$$C_{[2]} = C_{[1]} + p_{[2]} = p_{[1]} + p_{[2]}$$

$$\ldots$$

$$C_{[n-1]} = C_{[n-2]} + p_{[n-1]} = p_{[1]} + p_{[2]} + \ldots + p_{[n-1]}$$

$$C_{[n]} = C_{[n-1]} + p_{[n]} = p_{[1]} + p_{[2]} + \ldots + p_{[n]}.$$

By exploiting the above computations, we can rewrite the objective function as follows:

$$\sum_{j=1}^{n} C_j = np_{[1]} + (n-1)p_{[2]} + \ldots + 2p_{[n-1]} + p_{[n]}. \tag{2.3}$$

From (2.3) we can easily see that the contribution given to the objective function by the processing time of each job is inversely proportional to its position in the sequence; then, in order to minimize (2.3), it is enough to schedule the jobs in a nondecreasing order with respect to their processing time, following the rule (2.2). □

Remark 2.1.2. The SPT rule is used also to minimize the average completion time

$$\frac{1}{n} \sum_{j=1}^{n} C_j, \tag{2.4}$$

coinciding with (2.3) apart from the constant $1/n$. ||

Example 2.1.3. Assume to have five jobs ($n = 5$), whose processing times are the following: $p_1 = 3$, $p_2 = 5$, $p_3 = 1$, $p_4 = 7$ and $p_5 = 4$. As proved in Theorem 2.1.1, the optimal solution can be obtained by scheduling the jobs according to the sequence $\pi^* = (3, 1, 5, 2, 4)$.

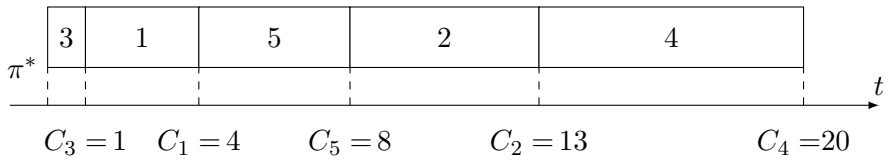

Figure 2.1: Example 2.1.3 - optimal schedule

The graphical representation of π^* is reported in Figure 2.1, from which we easily recover the optimal objective function value, which is

$$\sum_{j=1}^{n} C_j = C_1 + C_2 + C_3 + C_4 + C_5 = 46.$$

||

2.2 The Total Weighted Completion Time

Consider the following scheduling problem

$$1||\sum_{j=1}^{n} w_j C_j \tag{2.5}$$

that is a generalization of (2.1), with w_j, for $j = 1, \ldots, n$, representing the weight of job j.

Theorem 2.2.1. (Weighted Shortest Processing Time (WSPT) rule [48]) *The optimal solution to problem (2.5) is obtained by scheduling the jobs in a nondecreasing order with respect to the quantity* $\frac{p_j}{w_j}$*, i.e.*

$$\frac{p_{[1]}}{w_{[1]}} \leq \frac{p_{[2]}}{w_{[2]}} \leq \ldots \leq \frac{p_{[n]}}{w_{[n]}}. \tag{2.6}$$

Proof. Let π be a generic sequence violating rule (2.6) and let $f(\pi)$ be the corresponding objective function value. Then there exists a couple of indexes

(k, l), with $k = l - 1$, such that

$$\frac{p_k}{w_k} > \frac{p_l}{w_l},$$

or equivalently

$$w_k p_l - w_l p_k < 0. \tag{2.7}$$

Let π' be a sequence obtained from π by only swapping in π the positions of jobs k and l, and let Δ be the variation of the objective function obtained in passing from π to π'. We have:

$$\Delta = f(\pi') - f(\pi) = \sum_{j=1}^{n} w_j C'_j - \sum_{j=1}^{n} w_j C_j = w_k C'_k + w_l C'_l - w_k C_k - w_l C_l,$$

where C_j and C'_j, for $j = 1, \ldots, n$, are the completion times of job j in the sequences π and π', respectively. Since $C'_k = C_l$, we have

$$\Delta = w_k C_l + w_l C'_l - w_k C_k - w_l C'_k.$$

Moreover, taking into account that $C'_l - C'_k = -p_k$ and $C_l - C_k = p_l$, we obtain:

$$\Delta = w_l(C'_l - C'_k) + w_k(C_l - C_k) = -w_l p_k + w_k p_l < 0,$$

with the strict inequality following from (2.7).

Since $\Delta < 0$, we have shown that swapping in any sequence two successive jobs on the basis of rule (2.6) leads to an improvement of the objective function. Then the thesis follows, taking into account that the number n of jobs is finite. □

Example 2.2.2. Suppose we have four jobs ($n = 4$), whose processing times are $p_1 = 4$, $p_2 = 2$, $p_3 = 5$ and $p_4 = 7$ and whose weights are $w_1 = 4$, $w_2 = 6$, $w_3 = 1$ and $w_4 = 2$. The scenario is reported in Table 2.1.

From Theorem (2.2.1), the optimal sequence is $\pi^* = (2, 1, 4, 3)$, whose objective function value is recoverable from Figure 2.2 as follows:

$$\sum_{j=1}^{n} w_j C_j = w_1 C_1 + w_2 C_2 + w_3 C_3 + w_4 C_4 = 24 + 12 + 18 + 22 = 76.$$

||

j	p_j	w_j	p_j/w_j
1	4	4	1
2	2	6	1/3
3	5	1	5
4	7	2	7/2

Table 2.1: Example 2.2.2

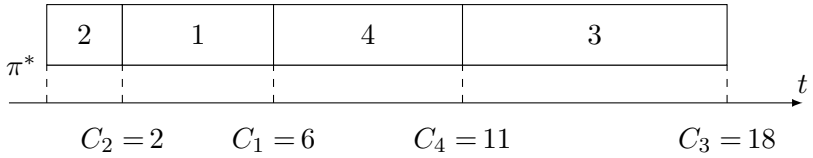

Figure 2.2: Example 2.2.2 - optimal schedule

2.3 The Total Completion Time with Release Times and Preemption

Consider the following scheduling problem:

$$1|prmp, r_j| \sum_{j=1}^{n} C_j. \tag{2.8}$$

An optimal solution to problem (2.8) is provided by Algorithm 1, where by $\bar{p}_j$ we denote the residual processing time of job j and by $\hat{J}$ the set of jobs that have still not been scheduled at the current iteration.

Algorithm 1: Solving problem (2.8)

Input: $J = \{1, \ldots, n\}$; $p_1, \ldots, p_n$; $r_1, \ldots, r_n$

▷Initialization

1 $r_{min} \leftarrow \min_{j \in J} r_j$
2 $r_{max} \leftarrow \max_{j \in J} r_j$
3 $T \leftarrow \sum_{j \in J} p_j + r_{max}$
4 $\hat{J} \leftarrow J$
5 **for** $j \leftarrow 1, \ldots, n$ **do**
6 $\quad \bar{p}_j \leftarrow p_j$
7 **for** $t \leftarrow r_{min}, \ldots, T$ **do**
$\quad$ ▷Identifying the available jobs
8 $\quad \bar{J} \leftarrow \{j \in \hat{J} \,|\, r_j \geq t\}$
$\quad$ ▷Computing the minimum residual processing time
9 $\quad j^* \leftarrow \arg\min_{j \in \bar{J}} \bar{p}_j$
$\quad$ ▷Scheduling
10 $\quad$ Process job j^* from time t to time $t+1$
$\quad$ ▷Updating the residual processing time
11 $\quad \bar{p}_{j^*} \leftarrow \bar{p}_{j^*} - 1$
$\quad$ ▷Updating the job set $\hat{J}$ to be scheduled
12 $\quad$ **if** $\bar{p}_{j^*} = 0$ **then**
13 $\quad\quad \hat{J} \leftarrow \hat{J} \setminus \{j^*\}$
$\quad$ ▷Stopping criterion
14 $\quad$ **if** $\hat{J} = \emptyset$ **then**
15 $\quad\quad$ **break**

Algorithm 1 can be summarized in the following steps. At any time t, among the jobs currently in the system and constituting the set $\bar{J}$ (line 8), we process job j^* having the lowest residual processing time (lines 9-10). Then, we update the residual processing time of j^* (step 11) and we iterate the procedure until all the jobs have been completely processed, i.e. all the residual processing times are equal to zero. From the practical point of view, once job j^* begins to be processed, its processing proceeds either until its completion, or until a new job with a lower processing time enters the system: in the latter case the processing of j^* is interrupted to be continued afterwards.

We do not prove the optimality of the solution provided by Algorithm 1,

but, on the other hand, we observe that the procedure is a sort of SPT rule applied to the set $\bar{J}$ of the currently available jobs.

We conclude the section by mentioning that, if preemption is not allowed, problem (2.8) becomes NP-hard. This is due to the fact that, at any time, we could start to process a very long job j, ignoring that a short job k can successively enter the system: then, since we cannot stop the current processing of j, we are forced to complete job j before job k, incurring in a low quality solution.

Example 2.3.1. Consider a scenario with three jobs ($n = 3$) whose processing times are $p_1 = 5$, $p_2 = 1$ and $p_3 = 6$, and whose the release times are $r_1 = 1$, $r_2 = 4$ and $r_3 = 5$.

Applying Algorithm 1, we start to process job 1 at $t = 1$ since at this time job 1 is the only available job. We continue its processing until $t = 4$, which is the time when job 2 enters the system. Now the set of the available jobs consists of jobs 1 and 2: since job 2 has the lowest (residual) processing time, we proceed by scheduling job 2 and interrupting the processing of job 1.

Continuing along the time horizon, at time $t = 5$ two events occur: job 2 ends its processing and leaves the system, while job 3 enters the system and it is now available for being processed. Then, at time $t = 5$, we have job 1 with residual processing time equal to 2 and job 3 with processing time equal to 6: as a consequence, we select job 1 which is entirely processed. Processing of job 3 ends the schedule.

In Figure 2.3 we represent the optimal solution, by highlighting the set of the available jobs together with their residual processing times. The optimal objective function value is:

$$\sum_{j=1}^{n} C_j = C_1 + C_2 + C_3 = 7 + 5 + 13 = 25.$$

||

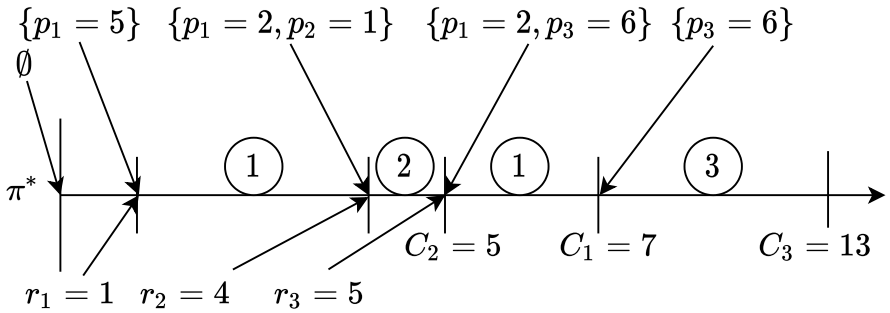

Figure 2.3: Example 2.3.1 - optimal schedule

2.4 The Maximum Lateness

Consider the following scheduling problem:

$$1||L_{max}. \tag{2.9}$$

Theorem 2.4.1. (Earliest Due Date (EDD) rule [27]) *An optimal solution to problem (2.9) can be obtained by processing the jobs in a nondecreasing order with respect to their due dates, i.e.:*

$$d_{[1]} \leq d_{[2]} \leq \ldots \leq d_{[n]}. \tag{2.10}$$

Proof. Since problem (2.9) is a particular case of problem (2.11), we refer the reader to the proof of Theorem 2.5.3, relative to problem (2.11).

□

Example 2.4.2. Consider the case of four jobs ($n = 4$), whose processing times are $p_1 = 3$, $p_2 = 6$, $p_3 = 2$ and $p_4 = 4$, and whose due dates are $d_1 = 3$, $d_2 = 6$, $d_3 = 1$ and $d_4 = 10$.

By Theorem 2.4.1 an optimal schedule is $\pi^* = (3, 1, 2, 4)$, whose a graphical representation is reported in Figure 2.4.

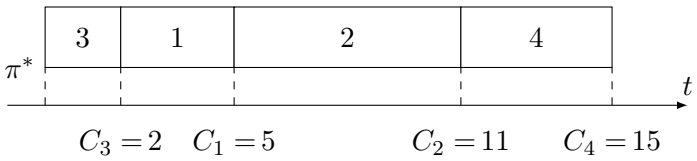

Figure 2.4: Example 2.4.2 - optimal schedule

In order to evaluate the objective function, we compute the lateness for each job:

- $L_1 = C_1 - d_1 = 5 - 3 = 2$;
- $L_2 = C_2 - d_2 = 11 - 6 = 5$;
- $L_3 = C_3 - d_3 = 2 - 1 = 1$;
- $L_4 = C_4 - d_4 = 15 - 10 = 5$.

Then, the optimal objective function value is

$$L^*_{max} = \max\{L_1, L_2, L_3, L_4\} = 5.$$

||

2.5 The Maximum Lateness with Precedence Constraints

Consider the following scheduling problem:

$$1|prec|L_{max}. \tag{2.11}$$

Problem (2.11) can be solved by Algorithm 2, which is a generalization of the EDD rule (2.10).

Algorithm 2: Solving problem (2.11)

Input: $J = \{1, \ldots, n\}$; $p_1, \ldots, p_n$; $d_1, \ldots, d_n$; $prec$
Output: $\pi^* = ([1], \ldots, [n])$

▷Initialization

1 $\hat{J} \leftarrow J$
2 $\bar{J} \leftarrow$ job set without successors in $\hat{J}$
3 **for** $t \leftarrow 1, \ldots, n$ **do**
4 $\quad j^* \leftarrow \arg\max_{j \in \bar{J}} d_j$
▷Scheduling
5 $\quad [n - t + 1] = j^*$
▷Updating the job set $\hat{J}$ to be scheduled
6 $\quad \hat{J} \leftarrow \hat{J} \setminus \{j^*\}$
▷Updating the job set $\bar{J}$
7 $\quad \bar{J} \leftarrow$ job set without successors in $\hat{J}$

The core of Algorithm 2 is constituted by lines 4-5, where at each iteration t job j^*, characterized by the maximum due date among the jobs without successors, is processed in the last available position $n - t + 1$.

Remark 2.5.1. Algorithm 2 is a particular case of the more general algorithm *Lowest Cost Last* (LCL rule) [42], solving the following problem:

$$1|prec|h_{max}, \tag{2.12}$$

where

$$h_{max} = \max\{h_1, \ldots, h_n\},$$

with h_j, for $j = 1, \ldots, n$, being a generic cost function, nondecreasing with respect to the completion time C_j.

At each iteration t, LCL schedules in the position $n - t + 1$ job j^* such that

$$j^* = \arg\min_{j \in \bar{J}} (h_j(\sum_{k \in \hat{J}} p_k)).$$

In case of problem (2.11), we have $h_j = L_j$ for $j = 1, \ldots, n$. ||

Remark 2.5.2. In case there are no precedence constraints, problem (2.11) reduces to problem (2.9) and Algorithm 2 coincides with the EDD rule (2.10). ||

Theorem 2.5.3. *Algorithm 2 provides an optimal solution to problem (2.11).*
Proof. Consider a sequence π obtained by Algorithm 2, in which, at a given iteration t among the jobs without successors, a job k, characterized by a higher due date with respect to j^*, is selected. This implies that, in π, j^* appears before k.

We can construct a new sequence π' obtained by moving, in π, job j^* immediately after job k. We note that all the jobs placed between j^* and k (included) in π are completed earlier in π'. The only job having an increase of its completion time is j^* that, in π', occupies the same position occupied by job k in π (position $n-t+1$). Recalling that at iteration t the job having highest due date is j^*, it is clear that

$$L_{[n-t+1]} > L'_{[n-t+1]},$$

where L and L' are the latenesses referred, respectively, to the sequences π and π'. Then, in passing from π to π', the objective function cannot increase. The proof follows taking into account that the number n of jobs is finite. □

Example 2.5.4. Consider a scenario with four jobs ($n = 4$) whose processing times are $p_1 = 2$, $p_2 = 4$, $p_3 = 3$ and $p_4 = 6$, while the corresponding due dates are $d_1 = 4$, $d_2 = 6$, $d_3 = 5$ and $d_4 = 1$. Suppose also to have the following precedence constraints:

$$1 \rightarrow 2,\ 4 \rightarrow 3,$$

imposing that job 1 must precede job 2 and job 4 must precede job 3. The scenario is represented in Table 2.2.

At the first iteration ($t = 1$), the job set $\bar{J}$ without successors contains jobs 2 and 3. Since $d_2 = \max\{d_2, d_3\}$, then $j^* = 2$ and consequently job 2 is

j	p_j	d_j	prec
1	2	4	2
2	4	6	-
3	3	5	-
4	6	1	3

Table 2.2: Example 2.5.4

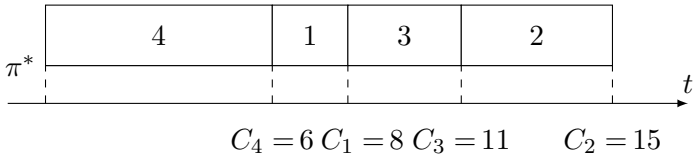

Figure 2.5: Example 2.5.4 - optimal schedule

processed in the last position of the sequence. Then the partial sequence is $\pi^* = (\cdot, \cdot, \cdot, 2)$.

At the second iteration ($t = 2$), $\bar{J} = \{1, 3\}$ and $d_3 = \max\{d_1, d_3\}$. Then job 3 is processed in the position $n - t + 1 = 3$, having $\pi^* = (\cdot, \cdot, 3, 2)$.

At this point, since all the precedence constraints are satisfied, the rule reduces to the simple EDD and then the final schedule (see Figure 2.5) is $\pi^* = (4, 1, 3, 2)$. In correspondence to π^*, the latenesses of the jobs are:

- $L_1 = C_1 - d_1 = 8 - 4 = 4$;
- $L_2 = C_2 - d_2 = 15 - 6 = 9$;
- $L_3 = C_3 - d_3 = 11 - 5 = 6$;
- $L_4 = C_4 - d_4 = 6 - 1 = 5$.

Then the optimal objective function value is

$$L^*_{max} = \max\{L_1, L_2, L_3, L_4\} = 9.$$

||

2.6 The Maximum Lateness with Release Times

Consider the following scheduling problem:

$$1|r_j|L_{max}. \tag{2.13}$$

We prove the NP-hardness of problem (2.13) [32], using a polynomial time reduction by the 3-Partition problem [18].

3-Partition problem. Given a set $A \stackrel{\triangle}{=} \{a_1, \ldots, a_{3t}\}$ of $3t$ positive integers, let b a positive integer such that $b/4 < a_j < b/2$, for $j = 1, \ldots, 3t$, and $\sum_{j=1}^{3t} a_j = tb$. Let $K \stackrel{\triangle}{=} \{1, 2, \ldots, 3t\}$ be the corresponding index set of A. Is there a partition of K into t subsets $K_1, \ldots, K_t$ such that $\sum_{j \in K_k} a_j = b$, for $k = 1, \ldots, t$? Note that each subset A_{K_k}, for $k = 1, \ldots, k$, must contain exactly three elements from A.

Theorem 2.6.1. *Problem (2.13) is NP-hard.*
Proof. The proof is based on a polynomial time reduction by the 3-Partition problem.

Given an instance (A, b) of the 3-Partition problem, with $A = \{a_1, \ldots, a_{3t}\}$, we construct an instance of problem (2.13), such that $L^*_{max} \leq 0$ if and only if set A can be partitioned into t triples having sum equal to b, as follows. We set

$$n = 4t - 1,$$

$$r_j = jb + (j-1), \quad p_j = 1, \quad d_j = jb + j, \quad \text{for } j = 1, \ldots, t-1$$

and

$$r_j = 0, \quad p_j = a_{j-t+1}, \quad d_j = tb + (t-1) \quad j = t, \ldots, 4t-1.$$

Observing that, for $j = 1, \ldots, t-1$, it holds $d_j = r_j + p_j$, we can conclude that $L^*_{max} \leq 0$ if and only if each job j, for $j = 1, \ldots, t-1$, starts its processing exactly at time r_j. This can be done if and only if the remaining jobs, whose processing times consist of the elements of the 3-Partitioning problem, can be partitioned over the t intervals of length b, i.e. if and only if the 3-Partitioning problem admits a solution. The idea is shown in Figure 2.6.

□

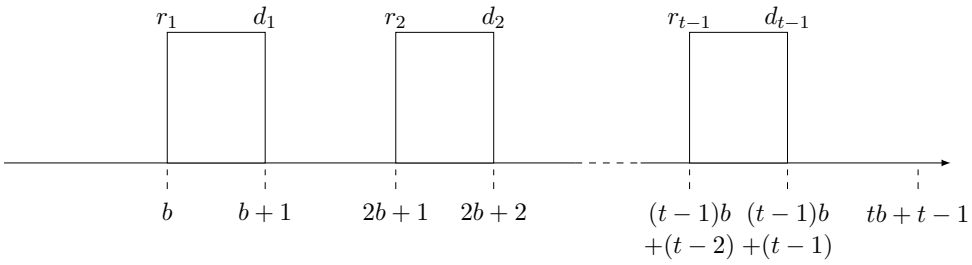

Figure 2.6

Chapter 3

Multiagent Scheduling Problems

In this chapter, we introduce a particular class of scheduling problems characterized by more than one decision maker (agent). This kind of problems involves different sets of jobs, one per agent, and generally each agent is interested in optimizing its own objective function, competing with the other ones for the usage of the shared resources (machines). Some cooperative cases, where all the agents contribute to the same global objective function(s), will be also discussed.

3.1 Introduction

In the previous chapter, we have seen some single-machine scheduling problems, where a decision maker, in order to optimize a given objective function, has to decide in which order to process a set of jobs on the machine.

Exploring a new area of the scheduling theory, we now focus on problems characterized by more than one decision maker. Since in the scheduling literature the decision makers are commonly named *agents*, this kind of problems are referred to as *multiagent* scheduling problems [3].

In a standard multiagent scheduling problem, each agent has its own job set, that contributes only to the specific agent's objective function: for this reason such problems are generally treated as multiobjective optimization problems [15], whose main difficulty resides in the fact that the agents have to share the same machines. On the other hand, there exist also some multia-

gent scheduling problems of the cooperative type, i.e. where the job sets of all the agents are involved in the same objective functions.

In the following, we provide the reader with some practical multiagent examples.

Example 3.1.1. (Rescheduling Problems)[4]) An example of such a problem with two agents is when two sets of jobs, one per agent, must be scheduled with the aim of minimizing the total flow time. The two sets of jobs differ in the fact that only one of them is characterized by deadlines: this is the case, for example, of jobs that have not been processed on a given day, becoming urgent on the next day. Then, on the next day, a rescheduling is needed. ||

Example 3.1.2. (Aircraft Landings [4]) In connection with Example 1.2.2, consider the assignment of the planes to the gates, which, in a cooperative type logic, is performed by exploiting an information exchange among the involved airlines. In this case, the multiagent nature of the problem comes out assuming that each airline is an agent that, in function of its flights, is interested in the maximization of the satisfaction of only its own passengers. ||

Example 3.1.3. (Project Scheduling [4]) Consider a company where different projects, each of them managed by a specific project manager, are to be scheduled at the same time. In such case each project manager is an agent, who has to compete with the other ones for the usage of shared resources (such as people and machinery), in order to perform the task related to her/his project. She/he is only interested in the performances of her/his project that, clearly, exclusively depends on the corresponding tasks. ||

Indicating by q the number of agents, a direct consequence of having $q > 2$ is that the schedule is in general evaluated with respect to more than one

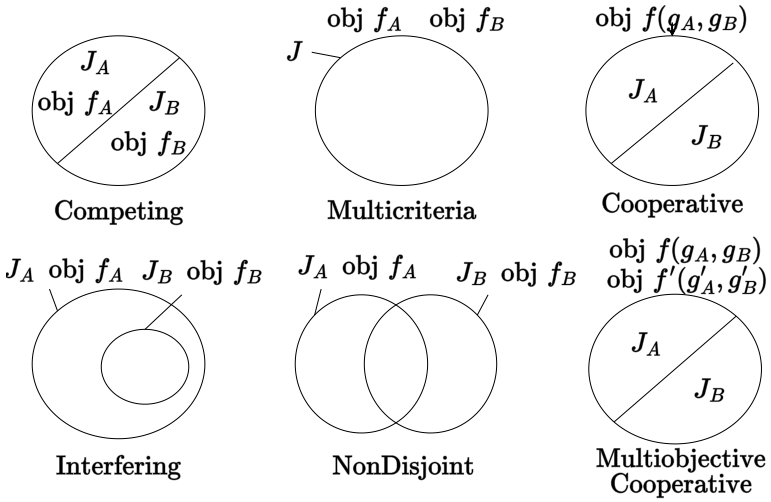

Figure 3.1: Possible scenarios for a multiagent scheduling problem in case of two agents ($q = 2$)

criterium, i.e. one per agent. Recalling that a generic schedule is denoted by π, we will use the notation $f_k(\pi)$, $k = 1, \ldots, q$, for referring to the kth criterion (we will assume that all the q criteria have to be minimized) and J_k for referring to the job set of agent k. In case $q = 2$, for the sake of simplicity, the two agents will be denoted by A and B, and the corresponding quantities by $\cdot_A$ and $\cdot_B$. As usual, the explicit dependence on π is in general omitted.

3.2 A Classification of the Multiagent Scheduling Problems

As mentioned above, in a multiagent scheduling problem all the agents share the same resources (machines), having different job sets (one per agent). Then, in the following, we report a possible classification (see Figure 3.1 when $q = 2$) of this kind of problems, based on the relationships which may exist among the various job sets, in connection with the involved criteria (objective functions).

3.2.1 Competing Agents

This situation, denoted by CO in the β field, represents a scenario in which the agents purely compete with each other for the usage of the resources. This happens when the agents have no job in common and they have different objective functions. Note that this scenario is symmetric, meaning that, given two different objective functions f and g, solving $1|CO|f_A, g_B$ is the same as solving $1|CO|g_A, f_B$.

3.2.2 Interfering Job Sets

In this scenario, denoted by IN in the β field, the job sets are nested and it is assumed that they are numbered is such a way that $J = J_1 \supseteq J_2 \supseteq \ldots \supseteq J_q$. This is not a symmetric case and, by focusing on the case with two agents, problems $1|IN|f_A, g_B$ and $1|IN|g_A, f_B$ often have different complexity.

3.2.3 Multicriteria Optimization

This scenario, denoted by MU (BI for two criteria) in the β field, is the typical multicriteria scheduling case. Here the agents share the same job set, i.e. $J_1 = J_2 = \ldots = J_q = J$, but they have different objective functions $f_1, f_2, \ldots, f_q$.

3.2.4 Nondisjoint Job Sets

In this situation, denoted by ND in the β field, each couple of agents may or may not share some jobs. An important aspect is that, for jobs belonging to more than one agent, the processing time is assumed to be only job dependent, while the other parameters may depend on the agent. For example, in case $q = 2$, a job $j \in J_A \cup J_B$ has a unique processing time p_j, but it may have two different due dates or two different weights (one per agent).

3.2.5 Cooperative Agents

Differently from the competing case, at the best of our knowledge, the cooperative multiagent scheduling problems have been treated in the literature only very recently.

In the cooperative case, all the agents (each of them having its own job set, as in the competing case), are interested in only one global objective function

f, which in turn is defined as a function of q different functions $g_1, \ldots, g_q$, one per agent. Then, while in the competing case the agents purely compete with each other for the usage of the shared resources, here all the jobs (each of them belonging to only one agent) contribute to the same (unique) objective function. The rescheduling problem, described in Example 3.1.1, falls in this scenario.

3.2.6 Multiobjective Cooperative Agents

This case is the same scenario as in Subsection 3.2.5: the only difference is that each agent contributes to more than one global objective function.

3.3 Solution Approaches

In this section, we briefly recall the standard solution methods adopted for solving a multiagent scheduling problem, reporting a more detailed description of the literature in Section 3.4.

Apart from the cooperative single-objective case described in Subsection 3.2.5 (Cooperative Agents), most of these problems involve more than one objective. Then, in what follows we recall, for the reader's convenience, the concepts of *strict* and *weak Pareto optimality.*

Definition 3.3.1. (**Strict Pareto optimality**) A schedule $\bar{\pi}$ is strict Pareto optimal with respect to the criteria $f_1, \ldots, f_q$, if there does not exist any schedule π such that $f_k(\pi) \leq f_k(\bar{\pi})$, for $k = 1, \ldots, q$, with at least one inequality strictly satisfied. The set of all the strict Pareto optimal solutions is named *Pareto set.* ||

Definition 3.3.2. (**Weak Pareto optimality**) A schedule $\bar{\pi}$ is weak Pareto optimal with respect to the criteria $f_1, \ldots, f_q$, if there does not exist any schedule π such that $f_k(\pi) < f_k(\bar{\pi})$, for $k = 1, \ldots, q$. ||

Remark 3.3.3. If a schedule is strict Pareto optimal, then it is also weak Pareto optimal. ||

Most of the approaches present in the literature are aimed at:

- computing a Pareto (usually strict) optimal schedule;
- computing the whole set of strict Pareto optimal schedules (this is generally done either iteratively or by means of a population algorithm);
- counting the number of the Pareto optimal schedules.

There exist also some techniques, named *feasibility techniques*, designed to solve a feasibility problem, i.e. a problem characterized by the absence of an objective function. In such case, the symbol _ is put in the γ field.

3.3.1 Linear Combination of Criteria

In case $q = 2$, this approach proposes to define a new objective function of the form $\alpha f_A + (1-\alpha) f_B$, where f_A and f_B are the criteria of agent A and agent B, respectively. The strategy is denoted by $\alpha f_A + (1-\alpha) f_B$ in the γ field. An important result [22] states that, by means of this approach, it is possible to compute only a subset of the Pareto set. This means that, in some cases, it is not possible to fix the value of α such that all the Pareto optimal solutions are retrieved.

3.3.2 Epsilon-Constraint Approach

The idea of this approach is to deal with many criteria, by optimizing one of them and by bounding the others.

In the case of two agents A and B (i.e. $q = 2$), if f_A is for example minimized, then f_B is imposed to be bounded by a constant Q (the role played by A and B is symmetric). In such case, the notation requires to insert f_A in the γ field and $f_B \leq Q$ in the β field (or the reverse in case the roles of A and B are swapped). It has been proved that this method outputs one weak Pareto optimal solution, whereas, for obtaining a strict Pareto optimal solution, it is necessary to solve a symmetric problem.

By means of this strategy, it is possible to generate the whole Pareto set by varying the values of Q within an interval $[L, U]$.

3.3.3 Lexicographic Order

This method consists in sorting the objective functions and in handling them on the basis of the obtained order.

In particular, let Π be the set of all the feasible schedules and let $f_1, f_2, \ldots, f_q$ be the q objective functions, in order, to be considered. The algorithms compute a sequence $\pi \in \Pi_q$ such that:

$$\Pi_1 = \{\pi \in \Pi : f_1(\pi) = \min_{\bar{\pi} \in \Pi}\{f_1(\bar{\pi})\}\},$$

$$\Pi_2 = \{\pi \in \Pi_1 : f_2(\pi) = \min_{\bar{\pi} \in \Pi_1}\{f_2(\bar{\pi})\}\},$$

$$\ldots$$

$$\Pi_q = \{\pi \in \Pi_{q-1} : f_q(\pi) = \min_{\bar{\pi} \in \Pi_{q-1}}\{f_q(\bar{\pi})\}\}.$$

3.3.4 Pareto Set Enumeration

This approach, denoted by $\mathcal{P}(X)$ in the γ field, has the aim of finding the whole Pareto set.

3.3.5 Counting

In this case, we are interested in counting the number of strict Pareto optimal solutions. In case $q = 2$, the notation, inserted in the γ field, is $\#(f_A, f_B)$.

3.4 Literature Review

The exploration of this new area of scheduling theory started in the first year of this century when, in [7, 11], the authors considered the possibility of involving more than one agent in the decision process.

The problems tackled in [11] are characterized by two or three agents and different combinations of the following three objectives have been considered: the minimization of maximum completion time, the minimization of the maximum lateness and the minimization of the weighted completion time. In [7] some two-agent scenarios have been faced, choosing as objective functions the maximum of regular functions (associated with each job), the number of

late jobs and the total weighted completion times. An extension of the latter problems to the case with more than two agents is reported in [8].

In the last years various works have focused on two-agent single-machine scheduling problems. In [6] a branch & bound algorithm has been designed, based on the computation of the bounds by means of a Lagrangian relaxation technique, with the aim of minimizing the total weighted completion time of the jobs of the first agent, subject to a bound on the objective function of the second agent, consisting of the alternative three cases: the minimization of the total weighted completion time, the minimization of the maximum lateness and the minimization of the maximum completion time. In [39] a 2-approximation algorithm has been presented to minimize the sum of two objectives, the maximum weighted completion time of the jobs of the first agent and the total weighted completion time of the jobs of the second agent. In [53] and [35] possible release times of the jobs have been taken into account. In particular, in [53] the objective is to minimize the tardiness of the first agent, keeping the lateness of the second one below a certain level, while in [35] the linear combination of the makespans of both the agents is minimized.

More recently, in [33] the authors, in a multitasking scenario, have addressed many two-agent problems by considering as cost function the maximization of regular function, the makespan, the maximum lateness and so on. While they have considered the possibility of having different objective functions among the agents, in [34] the authors have focused on a multiagent problem in which each agent is interested in the same objective function aimed at minimizing its total weighted late work.

Focusing on integrated-services packet-switched networks, in [51] the authors have come out with a multiagent scheduling problem. Their scenario considers several applications (agents) interested in the optimization of a unique criterion, which depends on the processing of their packets (jobs) on a shared common network (machine). While in the previous cases each agent was equipped with an objective function (competing scenario), here the aim is to minimize a unique cost function consisting in the sum of the makespan of all the agents (cooperative case).

In [55], considering a multiagent scheduling problem with release dates and preemption on a single machine, the authors have considered three different job structures: competing, nondisjoint and multicriteria. They have also studied different scheduling problems using, respectively, the epsilon-constraint method, the linear combination approach and the Pareto set enumeration

technique. They have proved that the first type of problems is polynomially solvable, while the other two are NP-hard, becoming polynomial if the number of scheduling criteria is fixed and the objective functions are *lateness-like*, such as L_{max} or T_{max}.

Part II

A New Scheduling Problem for Balancing the Average Weighted Completion Times

Chapter 4

Problem Definition and Computational Complexity

In this chapter, we introduce a new single-machine scheduling problem aimed at balancing the average weighted completion times of two classes of jobs. For such a problem, which is interpretable as a two-agent problem of the *cooperative* type, we prove the NP-hardness.

4.1 Problem Definition

We propose a new single-machine two-agent scheduling problem, aimed at balancing the average weighted completion times of two classes of jobs belonging to two agents, A and B, respectively. For the sake of brevity, in the sequel we refer to this problem as BAWCT-2 (Balancing the Average Weighted Completion Times of 2 agents).

Obtaining balanced solutions in a multiagent scenario reminds the concept of *fairness* [13], which plays an important role in case an optimal solution, maximizing the system utility, is unacceptable by the worse-off agent. Recent works in this field are [5, 10, 38, 57]: in particular, some of them concern the so-called *price of fairness* [12], which is a measure of how much the system utility has to be sacrificed in order to have a fair (balanced) solution.

Although BAWCT-2 is characterized by a *competing* scenario since the job sets of A and B do not intersect, this problem is apparently of the *cooperative* type as all the jobs of the two agents contribute to the same (unique) objective function.

Denoting by J_A and J_B the job sets of the agents A and B, respectively, and by n_A and n_B the corresponding cardinalities, using the Graham notation [25] the problem can be stated as:

$$1|\,|\left|\bar{C_A} - \bar{C_B}\right|, \tag{4.1}$$

where

$$\bar{C_A} = \sum_{j \in J_A} w_j C_j$$

and

$$\bar{C_B} = \sum_{j \in J_B} w_j C_j$$

are, respectively, the average weighted completion times of the two agents.

Throughout the thesis, referring to problem (4.1), we denote by C^* the optimal objective function value and by $C(\pi)$ the objective function value in correspondence to any job sequence π.

BAWCT-2 has been inspired by a real world problem arisen in an academic context and described in Example 4.2.1 of the next section.

4.2 Some Applications

In this section, we describe some examples characterized by the necessity of balancing the average completion times of two classes of jobs.

Example 4.2.1. (Exams at University) In a recovery exam session, a professor (the machine) has to schedule the oral examinations of two groups of students (the jobs) belonging to two different classes. Each student should be examined only on the parts of the syllabus where she/he failed during the regular exam session and then it is assumed that an estimate of the duration of the exam is known (and it is different) for everyone. Since the exam session is very short and, at the same time, there are a few available classrooms because of the lectures ongoing in the same period, he convokes all of them at the same time in the same classroom. If he would like to be impartial as much as possible toward the two classes, what is the best way to schedule the students with the aim of balancing the average waiting times of the two groups? ||

Remark 4.2.2. Example 4.2.1 can be modelled as problem (4.1), assuming that all the jobs (the students) have unitary weight. ||

Example 4.2.3. (Freight transportation) Consider a freight transportation company, performing deliveries by a single drone. The shipping times depend on the depot-customer distance and mainly on the load of the corresponding cargo (which can be very different among the various shipments), the latter acting on the speed of the drone. Imagine that, in a given time period, the company has to serve two customers, each of them requiring various freights of different types and loads. Assume moreover that the drone can perform only one delivery at a time and the company wants to balance the average waiting times of the customers. This problem could be modelled as problem (4.1), with the drone being the machine and the deliveries to the two customers being J_A and J_B: for any job j, the weight w_j is set equal to 1 and the processing time p_j is computed taking into account both the depot-customer distance and the load of the cargo. ||

Example 4.2.4. (Production of semi-finished products) Another example could be to balance the production of the semi-finished products, needed to assemble a finished product and characterized by high storage costs. Imagine the case where all the semi-finished products are manufactured by the same company owing a unique plant, which is able to perform in sequence different operations. Then all the semi-finished products are obtained by the same plant (single machine) performing in sequence, without precedence constraints, a finite number of independent different operations (jobs). Since the final product requires to assemble all the semi-products, in case the storage of the semi-products is expensive, a natural criterion to optimize the production could be to balance as much as possible the manufacture of the semi-products, by balancing the average completion times of the corresponding operations (more classes of jobs, one per semi-product), each of them characterized by a specific processing time. ||

4.3 Computational Complexity

In the sequel, we prove the NP-hardness of BAWCT-2 by means of a polynomial reduction from the well-known NP-complete partition problem [19].

Partition problem. Given a set of t positive integers $A \triangleq \{a_1, \dots a_t\}$, let $K \triangleq \{1, 2, \dots, t\}$ be the corresponding index set. Is there a partition of K into two subsets K_1 and K_2 such that $\sum_{j \in K_1} a_j = \sum_{j \in K_2} a_j$?

Theorem 4.3.1. *Problem (4.1) is NP-hard.*
Proof. Given an instance $A = \{a_1, \dots, a_t\}$ of the partition problem, we construct an instance of problem (4.1), by setting:

$$\begin{cases} J_A = \{1, \dots, t\}; & J_B = \{t+1\}; \\ p_j = a_j, \ j = 1, \dots, t; & p_{t+1} = 1; \\ w_1 = 1; & w_j = 0, \ j = 2, \dots, t; \\ w_{t+1} = \frac{\sum_{j=1}^{t} a_j}{2t(\sum_{j=1}^{t} a_j + 1)}. \end{cases} \tag{4.2}$$

We show that the partition problem has a solution if and only if, in correspondence to the instance (4.2), problem (4.1) admits an optimal sequence π^* with $C^* = C(\pi^*) = 0$.

($\Rightarrow$) Suppose that the partition problem admits a solution, i.e. there exists a partition of T into two subsets T_1 and T_2 such that $\sum_{j \in T_1} a_j = \sum_{j \in T_2} a_j$. Then

$$\sum_{j \in T_1} a_j = \sum_{j \in T_2} a_j = \frac{\sum_{j=1}^{t} a_j}{2}. \tag{4.3}$$

Without loss of generality, assume that $1 \in T_1$ and let $\bar{T}_1 \triangleq T_1 \setminus \{1\}$. In case $1 \in T_2$, it is sufficient to switch the roles played by T_1 and T_2. Then the sequence $\pi^* = (\bar{T}_1, 1, T_2, t+1)$, where the jobs in $\bar{T}_1$ and in T_2 are scheduled in any order, is an optimal solution to problem (4.1) with $C^* = 0$. In fact, taking into account (4.3) and recalling that $w_1 = 1$ and $w_j = 0$ for $j = 2, \dots, t$, we have

$$\bar{C}_A = \frac{\sum_{j \in J_A} w_j C_j}{n_A} = \frac{C_1}{t} = \frac{\sum_{j=1}^{t} a_j}{2t}$$

and

$$\begin{aligned}\bar{C}_B &= \frac{\sum_{j\in J_B} w_j C_j}{n_B} = w_{t+1}C_{t+1} \\ &= \frac{\sum_{j=1}^{t} a_j}{2t(\sum_{j=1}^{t} a_j+1)}(\sum_{j=1}^{t} a_j + 1) \\ &= \frac{\sum_{j=1}^{t} a_j}{2t}.\end{aligned}$$

Then $|\bar{C}_A - \bar{C}_B| = 0$ and, consequently, π^* is an optimal solution to problem (4.1).

($\Leftarrow$) Suppose that, in correspondence to the instance (4.2), problem (4.1) admits an optimal sequence π^* such that $C^* = 0$. Then

$$\bar{C}_A = \bar{C}_B,$$

i.e.

$$\frac{C_1}{t} = w_{t+1}C_{t+1}.$$

Substituting the value of w_{t+1} in the above equality, we obtain:

$$C_1 = \frac{C_{t+1}\sum_{j=1}^{t} a_j}{2(\sum_{j=1}^{t} a_j + 1)},$$

i.e.

$$C_{t+1} = 2C_1 + \frac{2C_1}{\sum_{j=1}^{t} a_j}. \tag{4.4}$$

Since the processing times in the instance (4.2) are integer, then all the completion times are integer. As a consequence also the second term of the right hand side of (4.4) is integer, i.e. there exists an integer $s \geq 0$ such that

$$2C_1 = s\sum_{j=1}^{t} a_j. \tag{4.5}$$

The case $s = 0$ cannot occur because $C_1 > 0$. Moreover, in case $s \geq 2$, from (4.5) we would have

$$C_1 \geq \sum_{j=1}^{t} a_j$$

and, from (4.4), it would follow:

$$C_{t+1} \geq 2\sum_{j=1}^{t} a_j + 2,$$

which is impossible since each completion time cannot exceed the sum of all the processing times, which is equal to $\sum_{j=1}^{t} a_j + 1$. Then in (4.5) we have necessarily $s = 1$, i.e.

$$C_1 = \frac{\sum_{j=1}^{t} a_j}{2}. \tag{4.6}$$

In summary, we have shown that, whenever $C^* = 0$ in correspondence to the instance (4.2), in any optimal sequence the completion time of job 1 is given by formula (4.6). As a consequence, comparing (4.3) and (4.6), a solution to the partition problem is obtained by setting T_1 (or indifferently T_2) equal to $\{1\} \cup \bar{T}_1$, where $\bar{T}_1$ contains all and only the indexes corresponding to the jobs which, in any optimal sequence, precede job 1. □

Chapter 5

Problem Formulation

In this chapter, we propose a mathematical formulation of BAWCT-2 as a nonsmooth variant of the well known quadratic assignment problem. Such formulation can be easily transformed into a mixed integer quadratically constrained optimization problem, which in turn reduces to a mixed integer linear program by applying the Glover linearization.

5.1 A Mixed Integer Quadratically Constrained Formulation

In its general form, problem (4.1) can be modelled as a nonsmooth variant of the well known quadratic assignment problem (see [14] for an extensive review), as follows.

We define, for $j \in J$ and $t = 1, \ldots, n$, the following decision variables:

$$x_{jt} \triangleq \begin{cases} 1 & \text{if job } j \text{ is assigned to position } t \\ 0 & \text{otherwise,} \end{cases}$$

grouped, for the sake of simplicity, into the vector x of dimension n^2.

Taking into account that the completion time of a job j scheduled in the position t is

$$p_j + \sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li},$$

problem (4.1) can be formulated as follows:

$$\begin{cases} C^* = \min\limits_{x} \quad \left| \dfrac{1}{n_A} \left[\displaystyle\sum_{j \in J_A} w_j \left(p_j + \sum_{t=1}^{n} \left(\sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \right) x_{jt} \right) \right] \right. \\ \qquad\qquad - \quad \left. \dfrac{1}{n_B} \left[\displaystyle\sum_{j \in J_B} w_j \left(p_j + \sum_{t=1}^{n} \left(\sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \right) x_{jt} \right) \right] \right| \\ \qquad\qquad \displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J \\ \qquad\qquad \displaystyle\sum_{j \in J} x_{jt} = 1 \quad t = 1, \ldots, n \\ \qquad\qquad x_{jt} \in \{0, 1\} \quad j \in J, \quad t = 1, \ldots, n, \end{cases} \tag{5.1}$$

where the constraints are the classical assignment constraints, which impose that each job must be assigned to exactly one position and each position to exactly one job.

Letting $p_A \stackrel{\triangle}{=} \sum_{j \in J_A} w_j p_j$ and $p_B \stackrel{\triangle}{=} \sum_{j \in J_B} w_j p_j$, problem (5.1) is easily rewritable as follows:

$$\begin{cases} C^* = \min\limits_{x} \quad \left| \dfrac{1}{n_A} \left(p_A + \displaystyle\sum_{j \in J_A} \sum_{t=1}^{n} w_j x_{jt} \sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \right) \right. \\ \qquad\qquad \left. - \dfrac{1}{n_B} \left(p_B + \displaystyle\sum_{j \in J_B} \sum_{t=1}^{n} w_j x_{jt} \sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \right) \right| \\ \qquad\qquad \displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J \\ \qquad\qquad \displaystyle\sum_{j \in J} x_{jt} = 1 \quad t = 1, \ldots, n \\ \qquad\qquad x_{jt} \in \{0, 1\} \quad j \in J, \quad t = 1, \ldots, n, \end{cases}$$

which is a sort of quadratic assignment problem, characterized by a nonsmooth objective function. By introducing the auxiliary variable v, it can be rewritten in the following equivalent form, corresponding to a mixed integer quadratically constrained program:

$$
\begin{cases}
C^* = \min\limits_{x,v} & v \\
& v \geq \dfrac{1}{n_A}\left(p_A + \displaystyle\sum_{j\in J_A}\sum_{t=1}^{n} w_j x_{jt} \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}\right) \\
& -\dfrac{1}{n_B}\left(p_B + \displaystyle\sum_{j\in J_B}\sum_{t=1}^{n} w_j x_{jt} \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}\right) \\
& v \geq \dfrac{1}{n_B}\left(p_B + \displaystyle\sum_{j\in J_B}\sum_{t=1}^{n} w_j x_{jt} \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}\right) \\
& -\dfrac{1}{n_A}\left(p_A + \displaystyle\sum_{j\in J_A}\sum_{t=1}^{n} w_j x_{jt} \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}\right) \\
& \displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J \\
& \displaystyle\sum_{j\in J} x_{jt} = 1 \quad t = 1,\ldots,n \\
& x_{jt} \in \{0,1\} \quad j \in J, \quad t = 1,\ldots,n.
\end{cases}
\tag{5.2}
$$

A possible way to treat the nonlinearities of problem (5.2) is to use the Glover linearization [23], as shown in the next section.

5.2 Glover Linearization

Theorem 5.2.1. *Letting*

$$b_t \stackrel{\triangle}{=} (t-1)P, \quad t = 1, \ldots, n,$$

with $P \stackrel{\triangle}{=} \sum_{j \in J} p_j$, *problem (5.2) is equivalent to the following mixed integer*

linear program:

$$
\begin{cases}
C^* = \min\limits_{x,v,z} v & (5.3)\\
v \geq \dfrac{1}{n_A}\left(p_A + \displaystyle\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}\right)\\
\quad - \dfrac{1}{n_B}\left(p_B + \displaystyle\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}\right) & (5.4)\\
v \geq \dfrac{1}{n_B}\left(p_B + \displaystyle\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}\right)\\
\quad - \dfrac{1}{n_A}\left(p_A + \displaystyle\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}\right) & (5.5)\\
b_t x_{jt} + \displaystyle\sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} - z_{jt} \leq b_t\\
\quad j \in J, \quad t = 1\ldots, n & (5.6)\\
z_{jt} \leq \displaystyle\sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} \quad j \in J, \quad t = 1, \ldots, n & (5.7)\\
z_{jt} \leq b_t x_{jt} \quad j \in J, \quad t = 1, \ldots, n & (5.8)\\
\displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J & (5.9)\\
\displaystyle\sum_{j\in J} x_{jt} = 1 \quad t = 1, \ldots, n & (5.10)\\
z_{jt} \geq 0 \quad j \in J, \quad t = 1, \ldots, n & (5.11)\\
x_{jt} \in \{0, 1\} \quad j \in J, \quad t = 1, \ldots, n. & (5.12)
\end{cases}
$$

Proof. Problem (5.2) can be easily rewritten as follows:

$$
\begin{cases}
C^* = \min\limits_{x,v,z} v \\
v \geq \dfrac{1}{n_A}\left(p_A + \displaystyle\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}\right) \\
-\dfrac{1}{n_B}\left(p_B + \displaystyle\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}\right) \\
v \geq \dfrac{1}{n_B}\left(p_B + \displaystyle\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}\right) \\
-\dfrac{1}{n_A}\left(p_A + \displaystyle\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}\right) \\
z_{jt} = x_{jt}\displaystyle\sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} \quad j \in J, \quad t = 1,\ldots,n \\
\displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J \\
\displaystyle\sum_{j\in J} x_{jt} = 1 \quad t = 1,\ldots,n \\
x_{jt} \in \{0,1\} \quad j \in J, \quad t = 1,\ldots,n,
\end{cases}
$$

where, for $j \in J$ and $t = 1 \ldots n$, the nonlinear constraints

$$z_{jt} = x_{jt}\sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}$$

impose that

$$x_{jt} = 1 \quad \Rightarrow \quad z_{jt} = \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} \tag{5.13}$$

and

$$x_{jt} = 0 \quad \Rightarrow \quad z_{jt} = 0. \tag{5.14}$$

Since

$$b_t = (t-1)P = \sum_{l \in J} \sum_{i=1}^{t-1} p_l, \tag{5.15}$$

it is easy to show that, in problem (5.3)-(5.12) the linear constraints (5.6)-(5.8) and (5.11) guarantee the satisfaction of both the implications (5.13) and (5.14).

□

Problem (5.3)-(5.12) constitutes the Glover linearization of problem (5.2). Such linearization is obtained by introducing $n^2 + 1$ continuous variables and $4n^2 + 2$ constraints. In passing, we recall that, in the case of a standard quadratic assignment problem characterized by positive cost coefficients in the objective function, the Glover linearization reduces to the Kaufmann and Broeckx linearization [28], which requires n^2 new continuous variables and $2n^2$ additional constraints.

We conclude this section by showing that, under mild assumptions, the optimal objective function value of the continuous relaxation of problem (5.3)-(5.12) provides a null lower bound.

Lemma 5.2.2. *Let*

$$\frac{p_B}{n_B} \geq \frac{p_A}{n_A}. \tag{5.16}$$

If

$$P \geq \frac{2(p_B n_A - p_A n_B)}{n_B(n-1)w_A}, \tag{5.17}$$

with $w_A \stackrel{\triangle}{=} \sum_{j \in J_A} w_j$*, then the optimal objective function value of the continuous relaxation of problem (5.3)-(5.12) is equal to zero.*

Proof. We show that the solution

$$\begin{cases} x_{jt} = 1/n & j \in J, \quad t = 1, \ldots, n; \qquad (5.18) \\ z_{jt} = \dfrac{2(t-1)(p_B n_A - p_A n_B)}{n(n-1)n_B w_A} & j \in J_A, \quad t = 1, \ldots, n; \qquad (5.19) \\ z_{jt} = 0 & j \in J_B, \quad t = 1, \ldots, n \qquad (5.20) \end{cases}$$

is feasible for the continuous relaxation of problem (5.3)-(5.12) and provides a null optimal value of the objective function v.

First of all, we trivially observe that vector x, defined by (5.18), satisfies the assignment constraints (5.9)-(5.10) and, in addition, $0 \leq x \leq e$, where e is the vector of ones. Moreover, by (5.16), we have $z_{jt} \geq 0$ for $j \in J$ and $t = 1, \ldots, n$.

Taking into account (5.15) and (5.18) and since $n \geq 2$, constraints (5.6) are satisfied by any nonnegative values of z_{jt}.

As for constraints (5.7) and (5.8), it is easy to observe that, by definition of b_t, they coincide in correspondence to the values $x_{jt} = 1/n$. Then, to show their satisfaction, it is sufficient to prove that

$$z_{jt} \leq b_t/n,$$

for $j \in J$ and $t = 1, \ldots, n$. For $j \in J_B$ and $t = 1, \ldots, n$, the above inequality is trivially satisfied because of the nonnegativity of b_t and n. As for the indexes $j \in J_A$ and $t = 1, \ldots, n$, combining (5.17) and (5.19) and recalling the definition of b_t, we have:

$$\begin{aligned} z_{jt} &= \frac{2(t-1)(p_B n_A - p_A n_B)}{n(n-1)n_B w_A} \\ &\leq \frac{(t-1)P}{n} = \frac{b_t}{n}. \end{aligned}$$

About the optimality, constraints (5.4) and (5.5), together with the objective function (5.3), guarantee that, in correspondence to any optimal solution,

it holds:

$$v = \left| \frac{1}{n_A} \left(p_A + \sum_{j \in J_A} \sum_{t=1}^{n} w_j z_{jt} \right) - \frac{1}{n_B} \left(p_B + \sum_{j \in J_B} \sum_{t=1}^{n} w_j z_{jt} \right) \right|.$$

Substituting in the above equality the values of z_{jt} given by (5.19) and (5.20), we have:

$$\begin{aligned} v &= \left| \frac{1}{n_A} \sum_{j \in J_A} \sum_{t=1}^{n} w_j \frac{2(t-1)(p_B n_A - p_A n_B)}{n(n-1) n_B w_A} + \frac{p_A}{n_A} - \frac{p_B}{n_B} \right| \\ &= \left| \frac{w_A}{n_A} \frac{n(n-1)}{2} \frac{2(p_B n_A - p_A n_B)}{n(n-1) n_B w_A} + \frac{p_A}{n_A} - \frac{p_B}{n_B} \right| \\ &= \left| \frac{p_B n_A - p_A n_B}{n_A n_B} + \frac{p_A}{n_A} - \frac{p_B}{n_B} \right| \\ &= 0. \end{aligned}$$

□

Remark 5.2.3. Lemma 5.2.2 holds also when condition (5.16) is not satisfied. In such a case, it is sufficient to swap the roles played by J_A and J_B and condition (5.17) becomes

$$P \geq \frac{2(p_A n_B - p_B n_A)}{n_A (n-1) w_B}, \tag{5.21}$$

where $w_B \stackrel{\triangle}{=} \sum_{j \in J_B} w_j$. ||

Remark 5.2.4. It is easy to verify that, in the unweighted case, i.e. when $w_j = 1$ for any $j \in J$, since $w_A = n_A$ and $P = p_A + p_B$, condition (5.17) is

automatically satisfied, provided that $n \geq 3$. This condition is apparently a mild assumption even in the weighted case, as confirmed by its satisfaction in correspondence to the more than one thousand randomly generated test problems, used for our numerical experiments (see Chapter 9). ||

Chapter 6

The Unweighted Case with Identical Jobs

In this chapter, we consider a simplified version of BAWCT-2, obtained by assuming the presence of identical jobs and unitary weights, for which we provide an exact linear time algorithm. We also present a further variant of the problem, aimed at considering more explicitly the cardinalities of the two job sets in the objective function.

6.1 Introduction

In this section, we formalize problem (4.1) assuming that all the involved jobs are identical and unweighted. Then, recalling that J_A and J_B represent the job sets of the two agents having, respectively, cardinalities n_A and n_B, and letting $J = J_A \cup J_B$ and $n = n_A + n_B$, we focus on the simplest case in which we assume that all the n jobs in J have the same processing time ($p > 0$) and unitary weight. Then we come out with the following optimization problem, referred in the sequel to as BACTI-2 (Balancing the Average Completion Times of

Identical jobs of 2 agents):

$$\begin{cases} C^*_{Id} \triangleq \min\limits_{\pi} & \left| \dfrac{\sum\limits_{j\in J_A} w_j C_j(\pi)}{n_A} - \dfrac{\sum\limits_{j\in J_B} w_j C_j(\pi)}{n_B} \right| \\ & \text{with:} \\ & p_j = p \quad j \in J, \\ & w_j = 1 \quad j \in J, \end{cases} \tag{6.1}$$

which can be interpreted as a kind of subset-sum problem [37]. In fact, since all the n jobs have the same processing time p, in correspondence to any sequence π we have:

$$\begin{aligned} & C_{[1]} = p, \\ & C_{[2]} = 2p, \\ & \vdots \\ & C_{[n]} = np. \end{aligned} \tag{6.2}$$

Then, in correspondence to any subset $\bar{I} \subseteq I = \{1, \ldots, n\}$, we have

$$\sum_{i\in\bar{I}} C_{[i]} = p \sum_{i\in\bar{I}} i.$$

As a consequence, problem (6.1) reduces to finding two disjoint subsets $\bar{I}_A$ and $\bar{I}_B$ of I, having cardinality n_A and n_B respectively, such that $\bar{I}_A \cup \bar{I}_B = I$ and the quantity

$$p \left| \frac{\sum\limits_{i\in\bar{I}_A} i}{n_A} - \frac{\sum\limits_{i\in\bar{I}_B} i}{n_B} \right|$$

is minimum.

6.2 An Exact Linear Time Algorithm

In this section we present an algorithm, named OptBACTI-2, that provides an optimal solution to problem (6.1) in linear time.

Given the job sets J_A and J_B, the algorithm works by first assigning to some appropriate positions of the optimal sequence all the jobs of one set, that

in the sequel we will call the *pioneer set*, and by successively accommodating the jobs of the other set into the remaining positions. The choice of the pioneer set is made in function of the parity of n_A and n_B. In particular, when n_A and n_B are both even or both odd, the sets J_A and J_B play a symmetric role and, in this case, we can choose indifferently one of them as the pioneer set. On the other hand, if the cardinalities n_A and n_B of the two sets are neither both odd nor both even, the role played by J_A and J_B is no more symmetric and the pioneer set must necessarily be the set whose cardinality is even.

Based on these considerations, without loss of generality, we state the algorithm assuming J_A as the pioneer set. In fact, whenever n_A is odd and n_B is even, it is sufficient to swap in the algorithm the role played by the respective sets J_A and J_B, letting J_B be the pioneer set.

We denote by I the set of the current available positions in the optimal sequence; moreover, to take into account the role played by the parity of n_A and n_B, we introduce the binary switch variable s having the following meaning:

$$\begin{cases} s = 0 & \text{if both } n_A \text{ and } n_B \text{ are odd numbers,} \\ s = 1 & \text{otherwise.} \end{cases}$$

The exact algorithm, providing an optimal sequence π^* to problem BACTI-2, is described in Algorithm 3, where, in correspondence to any index set S, the subroutine $random(S, k)$ returns and eliminates k random indexes from S.

Some observations on Algorithm 3 are in order. First of all, the termination of the algorithm is trivially guaranteed due to the finiteness of the job sets J_A and J_B. Moreover we observe that, because of the assumption that J_A is the pioneer set, the algorithm switches between the following two cases: i) both n_A and n_B are odd numbers ($s = 0$); ii) n_A is even ($s = 1$). These two cases differ uniquely for the passage to line 11, which accommodates a single job of J_A into the position $n/2$ and which is performed just once, only in the case i), where both n_A and n_B are odd and, consequently, n is even making this step well posed. Based on these considerations, lines 11-12 can be interpreted as the step aimed at making the cardinality of J_A even: in fact, after they are executed, the case i) reduces to the case ii).

Furthermore it is worth noting that the optimal solution provided by the algorithm is not uniquely determined, since it depends on some random choices. In particular, in addition to the selection of the couple of jobs in J_A (line 14 of the cycle 13-19) to be accommodated into the optimal sequence at the cur-

Algorithm 3: OptBACTI-2

Input: J_A, J_B
Output: $\pi^* = ([1], \ldots, [n])$

▷Initialization

1 $n_A \leftarrow |J_A|$
2 $n_B \leftarrow |J_B|$
3 $n \leftarrow n_A + n_B$
4 **if** *both* n_A *and* n_B *are odd numbers* **then**
5 $\quad s \leftarrow 0$
6 **else**
7 $\quad s \leftarrow 1$
8 $T \leftarrow \{1, \ldots, \lfloor n/2 \rfloor + s - 1\}$
9 $I \leftarrow \{1, \ldots, n\}$

▷Assigning a job of the pioneer set J_A

10 **if** $s = 0$ **then**
11 $\quad [n/2] \leftarrow random(J_A, 1)$
12 $\quad I \leftarrow I \setminus n/2$

▷Assigning iteratively two jobs of the pioneer set J_A

13 **repeat**
14 $\quad i, j \leftarrow random(J_A, 2)$
15 $\quad t \leftarrow random(T, 1)$
16 $\quad [t] \leftarrow i$
17 $\quad [n + 1 - t] \leftarrow j$
18 $\quad I \leftarrow I \setminus \{t, n + 1 - t\}$
19 **until** $J_A = \emptyset$

▷Assigning J_B

20 Assign arbitrarily all the jobs of J_B to the remaining positions available in I

rent iteration, the couple assignment is characterized also by the choice of the index t (line 15), which is performed $\lfloor \frac{n_A}{2} \rfloor$ times among $\lfloor \frac{n}{2} \rfloor + s - 1$ numbers without repetitions: consequently the number of obtainable different optimal solutions, in terms of positions assigned to J_A and J_B, is equal to the binomial coefficient

$$\left(\begin{array}{c} \lfloor \frac{n}{2} \rfloor + s - 1 \\ \lfloor \frac{n_A}{2} \rfloor \end{array} \right).$$

We will show the optimality of Algorithm 3 in the next section. To better clarify its behaviour, in the following we report two examples considering, respectively, the cases with $s = 0$ and $s = 1$ and recalling that $\bar{C}_A$ and $\bar{C}_B$ denote the average completion times of the jobs in J_A and J_B, respectively.

Example 6.2.1. Letting $J_A = \{1, 2, 3\}$ and $J_B = \{4, 5, 6, 7, 8\}$, since the cardinalities of both the two sets are odd, we can indifferently set J_A or J_B as the pioneer set. Choosing J_A, the performed steps are the following.

Initialization: $n_A := 3$, $n_B := 5$ and $n := 8$. Since both n_A and n_B are odd, we set $s := 0$. As a consequence $T := \{1, 2, 3\}$ and $I := \{1, 2, 3, 4, 5, 6, 7, 8\}$.

Assigning a job of the pioneer set J_A: Selecting from J_A the random job 1, we have $[4] = 1$. Then $J_A := \{2, 3\}$ and $I := \{1, 2, 3, 5, 6, 7, 8\}$.

Assigning iteratively two jobs of the pioneer set J_A: We select jobs $i = 2$ and $j = 3$. Choosing $t = 1$, we have $[1] = 2$ and $[8] = 3$. Then $J_A := \emptyset$, $T := \{2, 3\}$ and $I := \{2, 3, 5, 6, 7\}$.

Assigning J_B: Since $J_A = \emptyset$, we arbitrarily assign all the jobs of J_B to the remaining available positions in the set I and we stop.

Then an optimal sequence is the following:

$$\pi^* = (2, 4, 5, 1, 6, 7, 8, 3).$$

with

$$\bar{C}_A = \frac{p + 4p + 8p}{3} = \frac{13}{3}p$$

and

$$\bar{C}_B = \frac{2p + 3p + 5p + 6p + 7p}{5} = \frac{23}{5}p.$$

Consequently $C^*_{Id} = |\bar{C}_A - \bar{C}_B| = \frac{4}{15}p.$ ||

Example 6.2.2. Letting $J_A = \{1, 2, 3, 4\}$ and $J_B = \{5, 6, 7\}$, the pioneer set is necessarily the set J_A, since its cardinality is even. Then the performed steps are the following.

Initialization: $n_A := 4$, $n_B := 3$ and $n := 7$. Since n_A is even and n_B is odd, we set $s := 1$. As a consequence $T := \{1, 2, 3\}$ and $I := \{1, 2, 3, 4, 5, 6, 7\}$.

Assigning iteratively two jobs of the pioneer set J_A: Selecting from J_A the random jobs $i = 1$ and $j = 2$ and choosing $t = 1$, we have $[1] = 1$ and $[7] = 2$. Then $J_A := \{3, 4\}$, $T := \{2, 3\}$ and $I := \{2, 3, 4, 5, 6\}$. Now we select jobs $i = 3$ and $j = 4$. Choosing $t = 2$, we have $[2] = 3$ and $[6] = 4$. Then $J_A := \emptyset$, $T := \{3\}$ and $I := \{3, 4, 5\}$.

Assigning J_B: Since $J_A = \emptyset$, we arbitrarily assign all the jobs of J_B to the remaining available positions in the set I and we stop.

Then an optimal sequence is the following:

$$\pi^* = (1, 3, 5, 6, 7, 4, 2).$$

with

$$\bar{C}_A = \frac{p + 2p + 6p + 7p}{4} = 4p$$

and

$$\bar{C}_B = \frac{3p + 4p + 5p}{3} = 4p.$$

Consequently $C^*_{Id} = |\bar{C}_A - \bar{C}_B| = 0$. ||

Remark 6.2.3. Differently from the second example, in the first one the optimal solution does not provide a perfect balancing between the average completion times of the job sets J_A and J_B: we will show in fact (see next section, Lemma 6.3.4) that, whenever both n_A and n_B are odd, then $C^*_{Id} > 0$. ||

Remark 6.2.4. Apart from the case $s = 0$ where initially a job of J_A is placed in position $n/2$, if, at each iteration, the index t is chosen as the smallest one in the set T, the jobs i and j of J_A are assigned to the extreme positions of the optimal sequence, one at the beginning and the other one at the end:

consequently, at the last step of the algorithm (line 20), the jobs belonging to J_B are accommodated into the central positions. In other words, an optimal assignment of the pioneer set J_A is identified by the following set $\bar{I}_A$ of positions:

$$\bar{I}_A = \left\{1, 2, \ldots, \left\lfloor \frac{n_A}{2} \right\rfloor, \frac{n}{2}, n - \left\lfloor \frac{n_A}{2} \right\rfloor + 1, n - \left\lfloor \frac{n_A}{2} \right\rfloor + 2, \ldots, n\right\} \tag{6.3}$$

if $s = 0$, and

$$\bar{I}_A = \left\{1, 2, \ldots, \left\lfloor \frac{n_A}{2} \right\rfloor, n - \left\lfloor \frac{n_A}{2} \right\rfloor + 1, n - \bar{n}_A + 2, \ldots, n\right\} \tag{6.4}$$

if $s = 1$.

Based on (6.3) and (6.4), problem (6.1) is solvable in constant time; on the other hand, since in the practical applications a complete solution of the problem requires to assign explicitly each job to exactly one position, the complexity of the algorithm is, however, $O(n)$ (see next section, Theorem 6.3.5). ||

6.3 Optimality

In this section we prove that the solution provided by Algorithm 3 is an optimal solution to problem (6.1), assuming J_A as the pioneer set without loss of generality, as explained in the previous section. We also give a characterization of the optimal objective function value C^*_{Id}.

Even if the overall notation adopted throughout the work is reported at the beginning of the thesis, we recall that $\langle j \rangle$ denotes the position of job j.

We first show the following two lemmas.

Lemma 6.3.1. *Problem (6.1) is equivalent to the following problem:*

$$z^* \stackrel{\triangle}{=} \min_{\pi} \left| \sum_{j \in J_A} \langle j \rangle - \frac{n_A(n+1)}{2} \right|, \tag{6.5}$$

with

$$z^* = \frac{C^*_{Id} n_A n_B}{np}. \tag{6.6}$$

Proof. Multiplying the objective function of problem (6.1) by $n_A n_B$, we obtain

$$\min_{\pi} \left| n_B \sum_{j \in J_A} C_j - n_A \sum_{j \in J_B} C_j \right|. \tag{6.7}$$

By (6.2) we have:

$$\sum_{j \in J_A} C_j + \sum_{j \in J_B} C_j = p \sum_{j=1}^{n} j = \frac{pn(n+1)}{2}$$

and consequently:

$$\sum_{j \in J_B} C_j = \frac{pn(n+1)}{2} - \sum_{j \in J_A} C_j. \tag{6.8}$$

Substituting (6.8) in (6.7), we obtain:

$$\min_{\pi} \left| n_B \sum_{j \in J_A} C_j - \frac{n_A pn(n+1)}{2} + n_A \sum_{j \in J_A} C_j \right|,$$

i.e.

$$\min_{\pi} \left| n \sum_{j \in J_A} C_j - \frac{n_A pn(n+1)}{2} \right|,$$

which, dividing by n, becomes:

$$\min_{\pi} \left| \sum_{j \in J_A} C_j - \frac{n_A p(n+1)}{2} \right|,$$

or, equivalently:

$$\min_{\pi} \left| p \sum_{j \in J_A} \langle j \rangle - \frac{n_A p(n+1)}{2} \right|.$$

Finally, dividing by p, we obtain:

$$\min_{\pi} \left| \sum_{j \in J_A} \langle j \rangle - \frac{n_A (n+1)}{2} \right|.$$

As a consequence, relationship (6.6) holds. □

Remark 6.3.2. Problem (6.5) can be interpreted as a simplified version of problem (6.1), since it is independent on p and involves only the jobs of the pioneer set J_A. This consideration reflects the scheme of Algorithm 3, which works by first accommodating into the optimal sequence all the jobs of J_A and, successively, by arbitrarily assigning the jobs of J_B into the remaining positions. ||

Remark 6.3.3. As observed for problem (6.1), also the equivalent problem (6.5) appears as a particular case of the subset-sum problem. In fact, given the position set $I = \{1, \ldots, n\}$, problem (6.5) reduces to find a subset $\bar{I} \subset I$, of cardinality n_A, such that the function

$$\left| \sum_{i \in \bar{I}} i - \frac{n_A(n+1)}{2} \right|$$

is minimized. As a consequence, it can be interpreted as a subset-sum problem whose objective is to minimize the deviation of the sum of the elements in $\bar{I}$, with respect to the target $\frac{n_A(n+1)}{2}$. ||

Lemma 6.3.4. *If both n_A and n_B are odd, the objective function of problem (6.5) takes the form*

$$x + \frac{1}{2},$$

with x being a nonnegative integer number. As a consequence, $C^*_{Id} > 0$.

Proof. The objective function of problem (6.5) is

$$\left| \sum_{j \in J_A} \langle j \rangle - \frac{n_A(n+1)}{2} \right|.$$

Then, if both n_A and n_B are odd, n is even and $n_A(n+1)$ is odd; consequently, since the term

$$\sum_{j \in J_A} \langle j \rangle$$

is integer, the proof follows. Finally, taking into account (6.6), it holds $C^*_{Id} > 0$. □

Theorem 6.3.5. *Algorithm 3 provides an optimal solution to problem (6.1), assigning each job exactly to one position, in time $O(n)$.*

Proof. The calculation of the computational complexity is trivial. In fact, apart from the initialization step (lines 1-9), the core of the algorithm resides in the iterative assignment of the random couples of jobs from J_A (lines 14-18) and in the accommodation of J_B in the remaining positions (line 20), taking into account that the assignment of an initial random job of J_A (lines 11-12) is eventually performed only once in time $O(1)$. More specifically, in time $O(n_A)$ the algorithm executes $\lfloor n_A/2 \rfloor$ iterations to accommodate each time two jobs of J_A into the optimal sequence, and terminates by assigning in time $O(n_B)$ all the jobs of J_B to the remaining positions of the sequence.

In order to show the optimality, we treat separately the two cases $s = 0$ and $s = 1$, respectively.

Case 1) $s = 0$: both n_A and n_B are odd.

By Lemma 6.3.1, problem (6.1) is equivalent to problem (6.5). The quantity $\sum_{j \in J_A} \langle j \rangle$, which corresponds to the sum of the positions assigned to the jobs of J_A, is easily computable on the basis of the following considerations. Initially (lines 11-12) a job in J_A is scheduled in the position $n/2$, while successively (lines 14-18) at each iteration two jobs in J_A are scheduled, for $\frac{n_A-1}{2}$ times, in the positions t and $n+1-t$, respectively. As a consequence:

$$\begin{aligned} \sum_{j \in J_A} \langle j \rangle &= \frac{n}{2} + \frac{n_A - 1}{2}(t + n + 1 - t) \\ &= \frac{n_A(n+1) - 1}{2} \end{aligned} \tag{6.9}$$

Substituting (6.9) in the objective function of problem (6.5), we obtain

$$\left| \frac{n_A(n+1) - 1}{2} - \frac{n_A(n+1)}{2} \right| = \frac{1}{2},$$

which, by Lemma 6.3.4, is the optimal function value of problem (6.5).

Case 2) $s = 1$: n_A is even.

Analogously to the previous case, since at each iteration the assignment of the random couples of jobs from J_A (lines 14-18) is performed for $\frac{n_A}{2}$ times in the positions t and $n + 1 - t$ respectively, we have

$$\sum_{j \in J_A} \langle j \rangle = \frac{n_A(n+1)}{2}$$

and, by Lemma 6.3.1, taking into account the objective function of problem (6.5), it follows $z^* = 0$. □

Remark 6.3.6. Given C^*_{Id}, by using formula (6.6) to compute the objective function value of problem (6.5), in correspondence to the optimal solutions of Example 6.2.1 and Example 6.2.2, we obtain, as expected, $z^* = \frac{1}{2}$ and $z^* = 0$, respectively. ||

Finally, from the proof of Theorem 6.3.5 and taking into account relationship (6.6), the following result holds.

Corollary 6.3.7. *Given problem (6.1), if at least one between n_A and n_B is even then $C^*_{Id} = 0$, otherwise*

$$C^*_{Id} = \frac{np}{2n_A n_B}.$$

6.4 A Variant of the Problem

In this section we face a variant of BACTI-2, in order to take into account more explicitly the cardinalities of J_A and J_B in the objective function, with respect to the formulation (6.1).

In particular we tackle the following problem:

$$\begin{cases} C^*_{Id} \triangleq \min\limits_{\pi} & \left| \dfrac{n_A}{n} \displaystyle\sum_{j \in J_A} C_j(\pi) - \dfrac{n_B}{n} \sum_{j \in J_B} C_j(\pi) \right| \\ & \text{with:} \\ & p_j = p \quad j \in J \end{cases} \tag{6.10}$$

aimed at balancing the total weighted completion times of the two job sets, whose weights are the relative cardinalities n_A/n and n_B/n of J_A and J_B, respectively. It is easy to see that, analogously to problem (6.1), also problem (6.10) can be interpreted as a special case of the subset-sum problem.

For solving this problem we propose a mathematical programming formulation by introducing, as in problem (5.1), the following binary decision variables:

$$x_{jt} \triangleq \begin{cases} 1 & \text{if job } j \text{ is assigned to position } t \\ 0 & \text{otherwise,} \end{cases}$$

with $j \in J$ and $t = 1, \ldots, n$. Then, recalling (see (6.2)) that $C_{[t]} = tp$, for $t = 1, \ldots, n$, an optimization model solving problem (6.10) is the following:

$$\begin{cases} C^*_{\bar{I}d} = \min\limits_{x} & \left| \dfrac{n_A}{n} \displaystyle\sum_{j \in J_A} \sum_{t=1}^{n} tpx_{jt} - \dfrac{n_B}{n} \sum_{j \in J_B} \sum_{t=1}^{n} tpx_{jt} \right| \\ & \displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J \\ & \displaystyle\sum_{j \in J} x_{jt} = 1 \quad t = 1, \ldots, n \\ & x_{jt} \in \{0, 1\} \quad j \in J \quad t = 1 \ldots, n, \end{cases} \tag{6.11}$$

where the constraints are the standard assignment constraints.

The objective function of problem (6.11) is a convex nondifferentiable piecewise affine function. As usual, such a problem can be linearized by introducing an additional decision variable, say v, resulting in the following equivalent mixed integer linear program:

$$
\begin{cases}
C^*_{\bar{I}d} = \min\limits_{x,v} & pv \\
& v \geq \dfrac{n_A}{n} \displaystyle\sum_{j \in J_A} \sum_{t=1}^{n} t x_{jt} - \dfrac{n_B}{n} \sum_{j \in J_B} \sum_{t=1}^{n} t x_{jt} \\
& v \geq \dfrac{n_B}{n} \displaystyle\sum_{j \in J_B} \sum_{t=1}^{n} t x_{jt} - \dfrac{n_A}{n} \sum_{j \in J_A} \sum_{t=1}^{n} t x_{jt} \\
& \displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J \\
& \displaystyle\sum_{j \in J} x_{jt} = 1 \quad t = 1, \ldots, n \\
& x_{jt} \in \{0,1\} \quad j \in J \quad t = 1 \ldots, n,
\end{cases}
\tag{6.12}
$$

which is characterized by one real variable, n^2 binary variables and $2n + 2$ constraints.

We note that, although problem (6.12) is an assignment type problem, it cannot be solved as a linear program, since the total unimodularity of the constraints matrix is lost, due to the presence of the first two constraints introduced to linearize the objective function of problem (6.11). On the other hand we observe that, when n_A and n_B coincide, problem (6.10) reduces to problem (6.1) and, consequently, it can be solved in constant time by using formulas (6.3) and (6.4), or in linear time by means of Algorithm 3 which provides in addition the job-position assignments.

In the sequel we will show that, whenever the difference between the cardinalities n_A and n_B of the two job sets J_A and J_B, respectively, is sufficiently large, the optimal unique solution consists in simply accommodating at the end of the sequence all the jobs, in any order, belonging to the set with the smallest cardinality and, consequently, in assigning to the first positions all the jobs, in any order, of the set with the largest cardinality.

Without loss of generality, in the following we consider J_A, the pioneer set, as the job set with the largest cardinality and J_B as the job set with the smallest cardinality. We prove first the following lemma.

Lemma 6.4.1. *Let $\bar{\pi} = (J_A, J_B)$ be a sequence in which the jobs of J_A (in*

any order) precede all the jobs of J_B (in any order). Then

$$\frac{n_A}{n}\sum_{j\in J_A} C_j(\bar{\pi}) - \frac{n_B}{n}\sum_{j\in J_B} C_j(\bar{\pi}) \geq 0 \tag{6.13}$$

if and only if $n_B \leq \lfloor \bar{n}_A \rfloor$, with $\bar{n}_A \triangleq \dfrac{\sqrt{5n_A^2+6n_A+1}-n_A-1}{2}$.

Proof. By (6.2) we have:

$$\sum_{j\in J_A} C_j(\bar{\pi}) = \frac{pn_A(n_A+1)}{2} \tag{6.14}$$

and

$$\sum_{j\in J_B} C_j(\bar{\pi}) = \frac{pn(n+1)-pn_A(n_A+1)}{2}. \tag{6.15}$$

Substituting (6.14) and (6.15) in (6.13), we obtain:

$$\frac{pn_A^2(n_A+1)-pn_Bn(n+1)+pn_Bn_A(n_A+1)}{2n} \geq 0, \tag{6.16}$$

which, multiplying by the positive quantity $2n/p$, becomes:

$$n_A^2(n_A+1)-n_Bn(n+1)+n_Bn_A(n_A+1) \geq 0.$$

Recalling that $n = n_A + n_B$, the above inequality can be rewritten as follows:

$$n_A^3+n_A^2-n_B^3-2n_B^2n_A-n_B^2 \geq 0,$$

i.e.

$$(n_A+n_B)(n_A^2-n_An_B+n_A-n_B^2-n_B) \geq 0.$$

Dividing by $n = n_A + n_B > 0$, we obtain:

$$n_A^2-n_An_B+n_A-n_B^2-n_B \geq 0, \tag{6.17}$$

whose solution, with respect to n_B, is:

$$\frac{-\sqrt{5n_A^2+6n_A+1}-n_A-1}{2} \leq n_B \leq \frac{\sqrt{5n_A^2+6n_A+1}-n_A-1}{2}.$$

The thesis follows recalling that n_B is a positive integer number.

□

Theorem 6.4.2. *If* $n_B \leq \lfloor \bar{n}_A \rfloor$*, with*

$$\bar{n}_A \triangleq \frac{\sqrt{5n_A^2 + 6n_A + 1} - n_A - 1}{2},$$

then the sequences characterized by all the jobs of J_A *in the first positions (in any order) and all the jobs of* J_B *in the last positions (in any order) are the unique optimal solutions to problem (6.10), with*

$$C^*_{\bar{I}d} = p\frac{n_A^3 - n_B^3 + n_A^2 - n_B^2 - 2n_B^2 n_A}{2n}.$$

Proof. Let $\bar{\pi} = (J_A, J_B)$ be a sequence in which the jobs of J_A (in any order) precede the jobs of J_B (in any order) such that $n_B \leq \lfloor \bar{n}_A \rfloor$. First of all, we observe that all the sequences obtained from $\bar{\pi}$ by switching the positions between two jobs of J_A or between two jobs of J_B are characterized by the same objective function value provided by $\bar{\pi}$.

Then, denoting by $f(\pi)$ the objective function of problem (6.10), we show the optimality and the uniqueness by proving that, in correspondence to any other sequence $\hat{\pi} \neq \bar{\pi}$, obtained from $\bar{\pi}$ by switching the positions between a job of J_A and a job of J_B, it holds $f(\hat{\pi}) > f(\bar{\pi})$. In fact, letting

$$g(\pi) \triangleq \frac{n_A}{n} \sum_{j \in J_A} C_j(\pi),$$

and

$$h(\pi) \triangleq \frac{n_B}{n} \sum_{j \in J_B} C_j(\pi),$$

we have

$$f(\pi) = |g(\pi) - h(\pi)|.$$

Since $p > 0$, it holds:

$$g(\hat{\pi}) > g(\bar{\pi}) \tag{6.18}$$

and

$$h(\hat{\pi}) < h(\bar{\pi}). \tag{6.19}$$

By Lemma 6.4.1, we have:

$$f(\bar{\pi}) = g(\bar{\pi}) - h(\bar{\pi}). \tag{6.20}$$

From (6.20), taking into account inequalities (6.18) and (6.19), we obtain:

$$f(\hat{\pi}) = g(\hat{\pi}) - h(\hat{\pi}) > f(\bar{\pi}).$$

Consequently, $\bar{\pi}$ is optimal and

$$C^*_{\bar{I}d} = f(\bar{\pi}) = \frac{n_A}{n} \sum_{i \in J_A} C_i(\bar{\pi}) - \frac{n_B}{n} \sum_{i \in J_B} C_i(\bar{\pi}). \tag{6.21}$$

Substituting (6.14) and (6.15) in (6.21), and taking into account that $n = n_A + n_B$, we obtain:

$$C^*_{\bar{I}d} = p\frac{n_A^3 - n_B^3 + n_A^2 - n_B^2 - 2n_B^2 n_A}{2n}.$$

□

Remark 6.4.3. If $n_A \geq 2$, the value $n_B = \lceil \frac{n_A}{2} \rceil$ satisfies the inequality $n_B \leq \lfloor \bar{n}_A \rfloor$. ||

Chapter 7

A Lagrangian Relaxation Approach

In this chapter, for solving the general problem BAWCT-2, we propose a Lagrangian relaxation approach, that we apply to the Glover linearization of the mixed integer quadratically constrained formulation of the problem, presented in Chapter 5.

7.1 Introduction

The Glover linearization (5.3)-(5.12) of the mixed integer quadratically constrained problem (5.2) is a mixed integer linear program characterized by n^2 integer variables, $n^2 + 1$ continuous variables and $4n^2 + 2n + 2$ constraints. As a consequence, solving it exactly requires a remarkable effort from the computational point of view, especially for large values of n.

Then, for solving problem (4.1) we propose a Lagrangian relaxation approach (see the surveys [26, 17, 20]), based on relaxing all the constraints of problem (5.3)-(5.12), except the assignment ones and the positivity constraints on variables z_{jt}. The motivation is to design a heuristic algorithm requiring to solve, at each iteration, a linear assignment problem.

7.2 The Lagrangian Relaxation Problem

Dualizing constraints (5.4)-(5.8) of problem (5.3)-(5.12) and grouping all the corresponding Lagrangian multipliers into the vector $\lambda \in \mathbb{R}^{3n^2+2}$, the Lagrangian relaxation problem of the mixed integer linear program (5.3)-(5.12) is the following:

$$LR(\lambda) \begin{cases} f_{LR}(\lambda) \stackrel{\triangle}{=} \min\limits_{x,v,z} f(x,v,z) \\ \\ \sum\limits_{t=1}^{n} x_{jt} = 1 \quad j \in J \\ \\ \sum\limits_{j \in J} x_{jt} = 1 \quad t = 1, \ldots, n \\ \\ z_{jt} \geq 0 \quad j \in J, \quad t = 1, \ldots, n \\ \\ x_{jt} \in \{0,1\} \quad j \in J, \quad t = 1, \ldots, n, \end{cases}$$

where $\lambda \geq 0$ and

$$
\begin{aligned}
f(x,v,z) \stackrel{\triangle}{=} & \, v + \left[\frac{\alpha}{n_A}\left(\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt} + p_A\right)\right] \\
& - \left[\frac{\alpha}{n_B}\left(\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt} + p_B\right)\right] - \alpha v \\
& + \left[\frac{\beta}{n_B}\left(\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt} + p_B\right)\right] \\
& - \left[\frac{\beta}{n_A}\left(\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt} + p_A\right)\right] - \beta v \\
& + \sum_{j\in J}\sum_{t=1}^{n} \gamma_{jt}\left(b_t x_{jt} + \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} - z_{jt} - b_t\right) \\
& + \sum_{j\in J}\sum_{t=1}^{n} \delta_{jt}\left(z_{jt} - \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}\right) \\
& + \sum_{j\in J}\sum_{t=1}^{n} \epsilon_{jt}\left(z_{jt} - b_t x_{jt}\right),
\end{aligned}
$$

with α, β, γ_{jt}, δ_{jt}, ϵ_{jt}, $j \in J$ and $t = 1, \ldots, n$, being the Lagrangian multipliers (grouped in vector λ) corresponding to the constraints (5.4)-(5.8), respectively.

The above function f can be arranged in the following form:

$$f(x,v,z) = f_x(x) + f_v(v) + f_z(z) + \Delta,$$

where

$$
\begin{aligned}
f_x(x) \stackrel{\triangle}{=} & \sum_{j\in J}\sum_{t=1}^{n}\gamma_{jt}b_t x_{jt} + \sum_{j\in J}\sum_{t=1}^{n}\gamma_{jt}\sum_{l\in J}\sum_{i=1}^{t-1}p_l x_{li} \\
& - \sum_{j\in J}\sum_{t=1}^{n}\delta_{jt}\sum_{l\in J}\sum_{i=1}^{t-1}p_l x_{li} - \sum_{j\in J}\sum_{t=1}^{n}\epsilon_{jt}b_t x_{jt}, \\
f_v(v) \stackrel{\triangle}{=} & \; v(1-\alpha-\beta), \\
f_z(z) \stackrel{\triangle}{=} & \sum_{j\in J_A}\sum_{t=1}^{n} z_{jt}\left(\frac{w_j\alpha}{n_A} - \frac{w_j\beta}{n_A} - \gamma_{jt} + \delta_{jt} + \epsilon_{jt}\right) \\
& + \sum_{j\in J_B}\sum_{t=1}^{n} z_{jt}\left(\frac{w_j\beta}{n_B} - \frac{w_j\alpha}{n_B} - \gamma_{jt} + \delta_{jt} + \epsilon_{jt}\right)
\end{aligned}
$$

and

$$
\Delta \stackrel{\triangle}{=} \frac{p_A\alpha}{n_A} - \frac{p_B\alpha}{n_B} + \frac{p_B\beta}{n_B} - \frac{p_A\beta}{n_A} - \sum_{j\in J}\sum_{t=1}^{n} b_t\gamma_{jt}. \tag{7.1}
$$

As a consequence, the Lagrangian problem $LR(\lambda)$ is a separable program and, for any value of $\lambda \geq 0$, it can be solved by separately solving the following linear programs:

$$
LR_x(\lambda)\begin{cases}
f_{LR_x}(\lambda) \stackrel{\triangle}{=} \min\limits_{x} f_x(x) \\
\sum\limits_{t=1}^{n} x_{jt} = 1 \quad j \in J \\
\sum\limits_{j\in J} x_{jt} = 1 \quad t = 1,\ldots,n \\
x_{jt} \geq 0 \quad j \in J, \quad t = 1,\ldots,n
\end{cases},
$$

$$LR_v(\lambda) \begin{cases} f_{LR_v}(\lambda) \stackrel{\triangle}{=} \min\limits_{v} f_v(v) \\ \\ 0 \leq v \leq \bar{v}, \end{cases}$$

and

$$LR_z(\lambda) \begin{cases} f_{LR_z}(\lambda) \stackrel{\triangle}{=} \min\limits_{z} f_z(z) \\ \\ 0 \leq z_{jt} \leq b_t, \quad j \in J, \quad t = 1, \ldots, n. \end{cases}$$

Some comments about the above problems are in order. Problem $LR_x(\lambda)$ is a linear assignment problem, for which it is well known that constraints $x_{jt} \in \{0, 1\}$ can be substituted by constraints $x_{jt} \geq 0$.

In problem $LR_v(\lambda)$, since variable v represents the original objective function of problem (6.1), then such variable is nonnegative and bounded from above by any objective function value $\bar{v}$, computed in correspondence to any arbitrary job sequence in the set J.

In problem $LR_z(\lambda)$ any variable z_{jt}, $j \in J$ and $t = 1, \ldots, n$, is bounded from above by b_t, as a consequence of constraints (5.8) and (5.12).

It is worth noting that problems $LR_v(\lambda)$ and $LR_z(\lambda)$ are easily solvable by inspection. In fact, indicating by $v(\lambda)$ and $z_{jt}(\lambda)$, $j \in J$ and $t = 1, \ldots, n$, the respective optimal solutions, we have:

$$v(\lambda) = \begin{cases} 0 & \text{if } \alpha + \beta \leq 1 \\ \\ \bar{v} & \text{otherwise,} \end{cases}$$

$$z_{jt}(\lambda) = \begin{cases} 0 & \text{if } \dfrac{w_j(\alpha - \beta)}{n_A} \geq \gamma_{jt} - \delta_{jt} - \epsilon_{jt} \\ \\ b_t & \text{otherwise} \end{cases}$$

for $j \in J_A$ and $t = 1, \ldots, n$, and

$$z_{jt}(\lambda) = \begin{cases} 0 & \text{if } \dfrac{w_j(\beta - \alpha)}{n_B} \geq \gamma_{jt} - \delta_{jt} - \epsilon_{jt} \\ \\ b_t & \text{otherwise} \end{cases}$$

for $j \in J_B$ and $t = 1, \ldots, n$.

Then, for any $\lambda \geq 0$, we come out with:

$$f_{LR}(\lambda) = f_{LR_x}(\lambda) + f_{LR_v}(\lambda) + f_{LR_z}(\lambda) + \Delta, \tag{7.2}$$

which constitutes a lower bound on the optimal objective function value C^* of problem (5.3)-(5.12), i.e.

$$f_{LR}(\lambda) \leq C^*, \quad \text{for any } \lambda \geq 0.$$

We conclude the section by providing the following theorem, which gives a characterization of any lower bound $f_{LR}(\lambda)$ in case Lemma 5.2.2 holds.

Theorem 7.2.1. *If*

$$P \geq 2 \max \left\{ \frac{p_B n_A - p_A n_B}{n_B(n-1)w_A}, \frac{p_A n_B - p_B n_A}{n_A(n-1)w_B} \right\}, \tag{7.3}$$

then $f_{LR}(\lambda) \leq 0$ *for any* $\lambda \geq 0$.
Proof. Let

$$LD \left\{ \; f^*_{LR} \stackrel{\triangle}{=} \max_{\lambda \geq 0} f_{LR}(\lambda) \right.$$

be the Lagrangian dual of problem (5.3)-(5.12). Then, for any $\lambda \geq 0$, it holds

$$f_{LR}(\lambda) \leq f^*_{LR}.$$

Because of the integrality property (see [26, 17, 20]), f^*_{LR} coincides with the objective function value of the continuous relaxation of problem (5.3)-(5.12). By (7.3), taking into account Lemma 5.2.2 and Remark 5.2.3, such value is equal to zero. □

7.3 A Heuristic Lagrangian Algorithm

We have all the ingredients to design a Lagrangian relaxation algorithm for solving problem (4.1).

The core of our approach is to generate, at each iteration for any fixed nonnegative value of λ, a trial solution $x(\lambda)$ to problem (5.1), or equivalently to problem (5.3)-(5.12), by solving the linear assignment problem $LR_x(\lambda)$.

Since a feasible solution of the scheduling problem trivially requires only the job-position assignments, we do not need, differently from a general Lagrangian relaxation framework, a feasibility recovering procedure to adjust the current solution in case the relaxed constraints (5.4)-(5.8), aimed at guarantee the optimality, are not satisfied. Instead, once a trial solution is generated, we try to improve it by performing a local search (see Subsection 7.3.1).

In Algorithm 4, named LagrangianBAWCT-2, we summarize our Lagrangian heuristic approach, where by L and U we denote, respectively, the lower bound and the upper bound (the incumbent), available at the current iteration on the optimal objective function value C^*. Moreover, for any fixed value of λ, we indicate by $\pi(\lambda)$ the sequence generated by $x(\lambda)$, such that $[t] = j$ in correspondence to $x_{jt}(\lambda) = 1$. We denote also by $\partial f_{LR}(\lambda)$ the subdifferential of function f_{LR} at λ (see for example the fresh survey [21] on nonsmooth optimization) and by $\|\cdot\|$ the Euclidean norm.

The steps of the algorithm reflect a standard Lagrangian relaxation scheme aimed at shrinking, at each iteration, the interval $[L, U]$ containing C^*. We stop the iterative procedure either after a prefixed time limit, or when a maximum number $iter_{max}$ of iterations is reached, or when the difference between U and L becomes less than or equal to a certain threshold.

At line 7, to possibly decrease the incumbent, we perform a local search (see for example [1]), which will be object of the next subsection. The best sequence π_λ provided by the local searches constitutes a heuristic solution to problem (6.1).

At lines 18-19, we update the vector λ of the multipliers by a standard subgradient method [47], aimed at solving the nonsmooth Lagrangian dual problem LD and where the projection on the nonnegative orthant guarantees the nonnegativity of λ. As concerns the computation of a subgradient g of f_{LR} at λ, we take

$$g = \begin{bmatrix} g_\alpha \\ \\ g_\beta \\ \\ g_{\gamma_{jt}} \quad j \in J, \quad t = 1, \ldots, n \\ \\ g_{\delta_{jt}} \quad j \in J, \quad t = 1, \ldots, n \\ \\ g_{\epsilon_{jt}} \quad j \in J, \quad t = 1, \ldots, n \end{bmatrix},$$

Algorithm 4: LagrangianBAWCT-2

Input: a sequence $\bar{\pi}_{init}$; $\lambda \geq 0$
Output: a sequence $\bar{\pi}$

▷Initialization
$\bar{\pi} \leftarrow \bar{\pi}_{init}$
$\bar{v} \leftarrow C(\bar{\pi}_{init})$
$L \leftarrow -\infty$
$U \leftarrow \bar{v}$
repeat
  ▷Computing a trial solution
  Solve problem $LR_x(\lambda)$ to compute $x(\lambda)$ and $f_{LR_x}(\lambda)$
  ▷Local search
  Starting from $\pi(\lambda)$, compute π_λ by a local search
  ▷Updating the incumbent
  Compute $C(\pi_\lambda)$
  if $C(\pi_\lambda) < U$ **then**
    $U \leftarrow C(\pi_\lambda)$
    $\bar{\pi} \leftarrow \pi_\lambda$
  ▷Updating the lower bound
  Solve problem $LR_v(\lambda)$ to compute $f_{LR_v}(\lambda)$
  Solve problem $LR_z(\lambda)$ to compute $f_{LR_z}(\lambda)$
  Compute Δ `// see formula (7.1)`
  $f_{LR}(\lambda) \leftarrow f_{LR_x}(\lambda) + f_{LR_v}(\lambda) + f_{LR_z}(\lambda) + \Delta$
  if $f_{LR}(\lambda) > L$ **then**
    $L \leftarrow f_{LR}(\lambda)$
  ▷Updating the multipliers
  Compute $\tau > 0$ and $g \in \partial f_{LR}(\lambda)$
  $\lambda \leftarrow \max\{0, \lambda + \tau \frac{g}{\|g\|}\}$
until *a stopping criterion is satisfied*

where

$$g_\alpha = \frac{1}{n_A}\left(p_A + \sum_{j \in J_A}\sum_{t=1}^{n} w_j z_{jt}(\lambda)\right) - \frac{1}{n_B}\left(p_B + \sum_{j \in J_B}\sum_{t=1}^{n} w_j z_{jt}(\lambda)\right) - v(\lambda),$$

$$g_\beta = \frac{1}{n_B}\left(p_B + \sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}(\lambda)\right)$$

$$-\frac{1}{n_A}\left(p_A + \sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}(\lambda)\right) - v(\lambda),$$

$$g_{\gamma_{jt}} = b_t x_{jt}(\lambda) + \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}(\lambda) - z_{jt}(\lambda) - b_t,$$

$$g_{\delta_{jt}} = z_{jt}(\lambda) - \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}(\lambda)$$

and

$$g_{\epsilon_{jt}} = z_{jt}(\lambda) - b_t x_{jt}(\lambda),$$

with $x_{jt}(\lambda)$, $j \in J$ and $t = 1, \ldots, n$, $v(\lambda)$ and $z_{jt}(\lambda)$, $j \in J$ and $t = 1, \ldots, n$, being the optimal solutions of the respective relaxed subproblems $LR_x(\lambda)$, $LR_v(\lambda)$ and $LR_z(\lambda)$.

Finally the stepsize τ is the Polyak stepsize [43], given by:

$$\tau = \frac{f^*_{LR} - L}{\|g\|}. \tag{7.4}$$

Note that, in case Theorem 7.2.1 holds, f^*_{LR} is equal to zero, providing $\tau = -L/\|g\|$, with $L \leq 0$. Vice versa, in case f^*_{LR} is unknown, its value in formula (7.4) can be substituted by the incumbent or, because of the integrality property, can be computed by initially solving the continuous relaxation of problem (5.3)-(5.12).

7.3.1 The Local Search

A relevant role in Algorithm 4 is played by the step at line 7, where a local search is performed. The aim is to provide a sequence π_λ possibly better than the initial sequence $\pi(\lambda)$, by exploring a neighborhood of the trial solution $x(\lambda)$.

The local search we propose is detailed in Algorithm 5, where the subroutine $Swap(\pi, [i], [j])$ returns a sequence π_{ij} obtained from the sequence π by swapping jobs $[i]$ and $[j]$. In particular, for each of the $\frac{n(n-1)}{2}$ couples of

indices (i, j), with $i < j$, we check whether swapping jobs $[i]$ and $[j]$ results in the decrease of the current objective function value. If this is the case, we execute the exchange, we update the current objective function value and we iterate the process.

Algorithm 5: Local search

```
Input: sequence π(λ)
Output: sequence π_λ
π_λ ← π(λ)
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        π_ij ← Swap(π_λ, [i], [j])
        Δ ← C(π_λ) − C(π_ij)
        if Δ > 0 then
            π_λ ← π_ij
        end
    end
end
```

A remarkable consideration is that, in correspondence to any sequence π_{ij}, generated by swapping jobs $[i]$ and $[j]$ in an initial sequence π, there is no need to compute from scratch the objective function value, which would be expensive. In fact, denoting by C_A and C_B and by $C_{A_{ij}}$ and $C_{B_{ij}}$ the weighted completion times of the jobs in J_A and J_B provided, respectively, by the initial sequence π and by the generated sequence π_{ij}, it is possible to show that $C_{A_{ij}}$ and $C_{B_{ij}}$ can be determined in function of C_A and C_B, by means of the following formulae:

$$\begin{aligned} C_{A_{ij}} &= C_A + \sum_{k \in J_{A_k}} w_{[k]} \left(p_{[j]} - p_{[i]} \right) \\ &+ \sum_{k \in J_{A_i}} w_{[i]} p_{[k]} - \sum_{k \in J_{A_j}} w_{[j]} p_{[k]} \end{aligned} \tag{7.5}$$

and

$$\begin{aligned} C_{B_{ij}} &= C_B + \sum_{k \in J_{B_k}} w_{[k]} \left(p_{[j]} - p_{[i]} \right) \\ &+ \sum_{k \in J_{B_i}} w_{[i]} p_{[k]} - \sum_{k \in J_{B_j}} w_{[j]} p_{[k]}, \end{aligned} \tag{7.6}$$

where

$$J_{A_k} \triangleq \begin{cases} \{k \mid i < k < j\} & \text{if } [k] \in J_A \\ \\ \emptyset & \text{otherwise,} \end{cases}$$

$$J_{A_i} \triangleq \begin{cases} \{k \mid i < k \leq j\} & \text{if } [i] \in J_A \\ \\ \emptyset & \text{otherwise,} \end{cases}$$

$$J_{A_j} \triangleq \begin{cases} \{k \mid i \leq k < j\} & \text{if } [j] \in J_A \\ \\ \emptyset & \text{otherwise} \end{cases}$$

and

$$J_{B_k} \triangleq \begin{cases} \{k \mid i < k < j\} & \text{if } [k] \in J_B \\ \\ \emptyset & \text{otherwise,} \end{cases}$$

$$J_{B_i} \triangleq \begin{cases} \{k \mid i < k \leq j\} & \text{if } [i] \in J_B \\ \\ \emptyset & \text{otherwise,} \end{cases}$$

$$J_{B_j} \triangleq \begin{cases} \{k \mid i \leq k < j\} & \text{if } [j] \in J_B \\ \\ \emptyset & \text{otherwise.} \end{cases}$$

Although using (7.5) and (7.6) would however require a linear time in n, it is worth noting that, since such formulae involve only the jobs placed between positions i and j, avoiding the recalculation of the objective function from the beginning leads to a relevant improvement on the performance of the overall algorithm.

Chapter 8

A Genetic Algorithm

In this chapter, we propose a genetic algorithm for solving the general problem BAWCT-2 described in Chapter 4. The aim of this approach, with respect to the Lagrangian relaxation technique proposed in Chapter 7, is to speed up the resolution process in order to face large scale problems.

8.1 Introduction

Genetic algorithms (GAs) are metaheuristic search approaches, successfully applied to solve different varieties of NP-hard optimization problems (see [30]).

In particular, given an optimization problem, they are based on the following evolution paradigm. One starts from an initial population where to each individual a genotype, representing a possible solution to the problem, is associated. The quality of such solution is measured by its fitness value, obtained by evaluating the so-called fitness function and expressing how well the solution fits the problem. In passing from a generation to the successive one, the aim is to possibly increase the overall fitness of the population, by generating, via specific genetic operators, new individuals (offspring) characterized by better fitness values.

The basic schema of a genetic algorithm is reported in Algorithm 6.

Even if in the literature there are plenty of strategies to implement the steps of Algorithm 6, each of such strategies can be in general adapted and tailored to the specific case, making GAs really attractive for solving many optimization problems. Some examples are [31] and [45], where the well-known traveling

Algorithm 6: Genetic algorithm

```
1  Pop ← GenerateInitialPopulation()
2  repeat
3  |  NewPop ← ∅
4  |  repeat
5  |  |  parents ← Selection(Pop)
6  |  |  offspring ← Crossover(parents)
7  |  |  offspring ← Mutation(offspring)
8  |  |  NewPop ← NewPop ∪ offspring
9  |  until NewPop is complete
10 |  Pop ← UpdatePop(Pop,NewPop)
11 until a stopping criterion is satisfied
```

salesman problem has been faced, and [2, 16, 24, 36, 40, 41, 46, 49, 54, 56], where GAs have been used for solving scheduling problems.

Before describing in detail how each step of Algorithm 6 has been implemented for solving BAWCT-2, it is worth specifying that in our approach, at each iteration, the previous population is completely replaced by the new one, maintaining only the fittest current individual in order to preserve a monotonic behavior of the best current fitness value (line 10).

8.2 Encoding and Fitness Evaluation

The first choice in the implementation of a genetic algorithm regards the encoding of each individual, representing a feasible solution to the problem. Since we are dealing with a single machine scheduling problem, in such case the standard way to encode an individual I is to use a one-dimensional array V_I, such that $V_I[t] = j$ if and only if the job j is scheduled in the position t. As a consequence, according to this notation, the job processed in the position t, as usual, will be denoted by $[t]$.

Another key point characterizing GAs is the definition of the fitness function, aimed at providing the fitness value of an individual. Since individuals with higher fitness are preferred by GAs and, on the other hand, BAWCT-2 is a minimization problem (see (4.1)), we propose to use the following fitness

function:

$$fitness_I = \frac{1}{f(I)}, \tag{8.1}$$

where $f(I)$ is the objective function value of problem (4.1) in correspondence to the solution represented by the individual I.

Note that using formula (8.1) ensures to highlight even slight differences in function f and, additionally, when $f(I) = 0$ (corresponding to a perfect balanced optimal solution), it trivially follows that $fitness_I \to +\infty$.

8.3 Initial Population

Once the encoding strategy has been defined, the next step to be performed in GAs is the generation of the initial population. A review of the main techniques used in this phase is reported in [29].

For solving BAWCT-2, we have tested the following three different techniques: *Random generation*, *Alternated generation* and *Bidirectional generation*. If, on one hand, using the *Random generation* strategy could generate strongly unbalanced solutions due to the completely random generation of the population, on the other hand the *Alternated generation* and the *Bidirectional generation* techniques should hopefully generate better starting points, since they are based on more elaborate criteria. In particular, while the *Alternated generation* technique has been used for generating a starting point in the Lagrangian relaxation approach proposed in [9], the *Bidirectional generation* is inspired by the strategy at the basis of the exact algorithm OptBACTI-2 (Algorithm 3), providing an optimal solution to the unweighted case with identical jobs.

In what follows, we formally describe the three strategies according to the above proposed encoding.

- *Random generation*. The genotype of each individual I, corresponding to a feasible solution of BAWCT-2, is given by a random permutation $\sigma(J)$ of J. In other words, $[i] = \sigma(J)_i$ for $i = 1, \ldots, n$.

- *Alternated generation*. The genotype of each individual I is given by alternating the jobs of the two classes until one of the two sets is completely scheduled. The remaining jobs, if any, are accommodated at the end of the sequence.

More formally, let $\sigma(J_A)$ and $\sigma(J_B)$ be two random permutations of J_A and J_B, respectively. Then, for each $i = 1, \ldots, \min\{n_A, n_B\}$, we initially set

$$[2i-1] = \sigma(J_A)_i \text{ and } [2i] = \sigma(J_B)_i.$$

Successively, in case $n_A \neq n_B$, the remaining jobs of the largest class are assigned at the end of the sequence, starting from the position $2 \cdot \min\{n_A, n_B\} + 1$.

- *Bidirectional generation.* The genotype of each individual is provided by an iterative procedure, which consists in accommodating, at each iteration, two jobs of one class (alternately chosen between A and B) in the first and in the last currently available positions of the sequence, respectively.

  In particular, let $\sigma(J_A)$ and $\sigma(J_B)$ be two random permutations of J_A and J_B, respectively. Then, at each iteration k, for $k = 0, \ldots, \lfloor n_A/2 \rfloor + \lfloor n_B/2 \rfloor - 1$, we set either

$$[k+1] = \sigma(J_C)_t \quad \text{and} \quad [n-k] = \sigma(J_C)_{t+1} \tag{8.2}$$

  or

$$[k+1] = \sigma(J_C)_{t+1} \quad \text{and} \quad [n-k] = \sigma(J_C)_t, \tag{8.3}$$

  where J_C is the index set coinciding, alternately at each iteration, with either J_A or J_B. The index t is initialized to 1 and it is increased by 2 at the end of each iteration.

  Obviously, when only one of the two classes is completely scheduled, J_C remains fixed to the index set of the other class. Moreover, in case n_A and (or) n_B are odd, the remaining jobs are put in the middle of the sequence, using a greedy strategy based on the evaluation of the fitness function. The fitness value is also considered in the final choice of the configuration between (8.2) and (8.3).

It is worth noting that, for all the above proposed techniques, the probability of getting two times the same schedule as output of the generation process is very low, since they are defined in function of random permutations. This is a crucial point, as it guarantees a *diversification* in the initial population.

8.4 Selection

Selection (see [52]) is a crucial step in designing GAs, since it defines the chance of a given individual to participate in the reproduction process. Considering that a convergence to optimal solutions is desired, it is recommended to give a major chance to individuals characterized by high fitness values.

In this work, three different well-known selection techniques have been adopted: *Roulette wheel*, *Binary tournment* and *k-tournment.*

- *Roulette wheel.* The selection is performed by simulating a roulette wheel, in which the chance of an individual to be selected is represented by a portion of the roulette. In particular, for each individual I, since the size of its portion must be proportional to the fitness value, the probability p_I to be selected is set as

$$p_I = \frac{fitness_I}{\sum_{i \in Pop} fitness_i},$$

  where Pop represents the overall population.

- *Binary tournament.* Between two randomly selected individuals, the one characterized by the highest fitness value is chosen for reproduction.

- *k-tournament.* It is the same as the *Binary tournament*, apart from the fact that k individuals, with $2 < k \leq p_{size}$, are involved in the tournament, where p_{size} is the size of the population.

8.5 Crossover

Once the parents are selected, they have to reproduce in order to generate new children (offspring). The reproduction process is simulated by using crossover operators, aimed at creating new individuals whose features are a mix of the genetic properties of the parents. From the algorithmic point of view, this phase is very important since it allows to explore the solution space.

In the literature, a lot of crossover operators have been proposed for scheduling problems and, in general, for sequencing problems, like the traveling salesman problem (for a general overview see [44] and [50]).

In [54], the authors have highlighted that the crossover operators, mainly proposed for the traveling salesman problem, do not perform very well in a

scheduling context. As a consequence, taking into account this consideration, for solving BAWCT-2 we have chosen to implement the following three crossover strategies: *One-point crossover*, *Two-point crossover* and *Position-based crossover*.

Given two parents I_1 and I_2, these operators can be described as follows:

- *One-point crossover.* An index $i \in [1, n]$ is randomly selected. Then, with the same probability, either the first i jobs or the last $n-i$ ones are inherited from I_1, while the remaining ones are inserted in the sequence preserving the order they have in I_2 (see Figure 8.1).

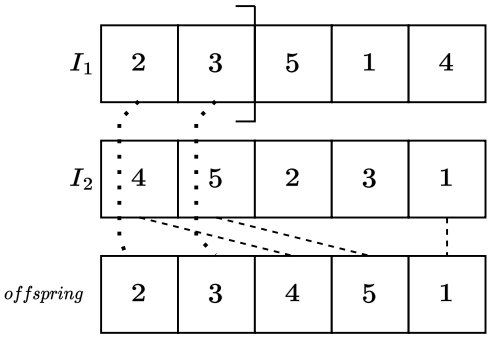

Figure 8.1: One-point crossover with $i = 2$.

- *Two-point crossover.* Two indices $i, j \in [1, n]$ are randomly selected. Assuming $i \leq j$, the first i jobs and the last $n - j$ ones are inherited from I_1, while the remaining $j-i$ are inserted in the sequence preserving the order they have in I_2 (see Figure 8.2).

- *Position-based crossover.* An index $k \in [1, n]$ is randomly selected and k different positions v_i, $i = 1, \ldots, k$, are sampled in the same interval $[1, n]$. Then the jobs in positions v_i, for $i = 1, \ldots, k$, are inherited from I_1, while the remaining ones are inserted in the sequence preserving the order they have in I_2 (see Figure 8.3).

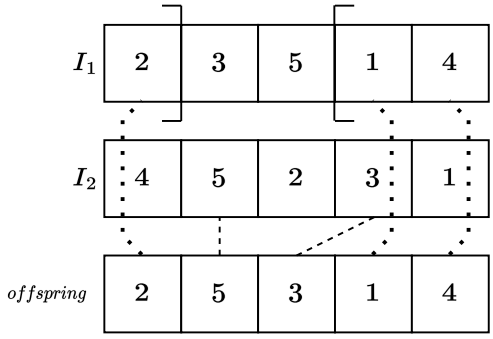

Figure 8.2: Two-point crossover with $i = 1$ and $j = 3$.

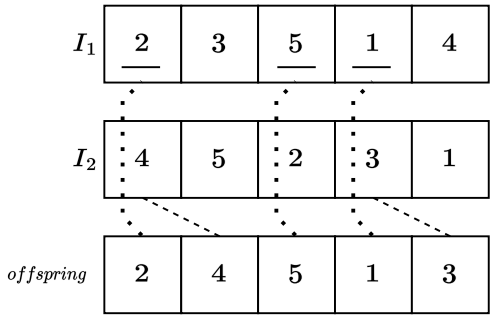

Figure 8.3: Position-based crossover with $k = 3$, $v_1 = 1, v_2 = 3$ and $v_3 = 4$.

8.6 Mutation

Another important operator in GAs is the mutation that, together with the crossover operator, allows to explore the solution space. In particular, the mutation operator is aimed at perturbing the current solution by applying random changes. Inspired by [54], in this work the following mutation operators have been tested:

- *Arbitrary two-job change.* Two positions $i, j \in [1, n]$ are randomly selected and the jobs currently scheduled in these positions are swapped (see Figure 8.4).

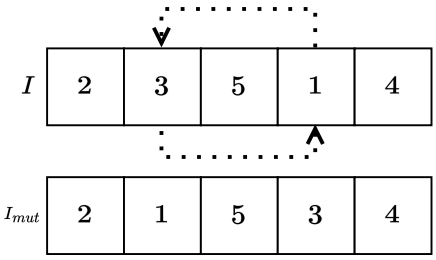

Figure 8.4: Arbitrary two-job change with $i = 2$ and $j = 4$.

- *Shift change.* Two positions $i, j \in [1, n]$ are randomly selected and the job in the position i is moved to the position j, shifting all the intermediate jobs (see Figure 8.5).

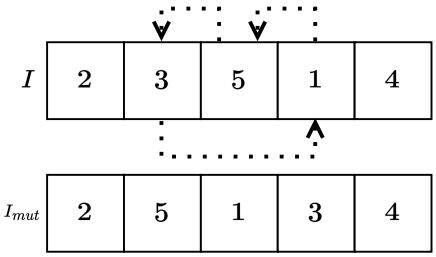

Figure 8.5: Shift change with $i = 2$ and $j = 4$.

Additionally, considering any set of adjacent jobs as a *batch*, we have also defined the following mutation operator:

- *Arbitrary batch change.* Given two random positions $i, j \in [1, n]$, assume $i \leq j$. Then an integer s (the batch size) is randomly determined in the interval $[1, \min\{j - i, n + 1 - j\}]$ and, for $k = 0, \ldots, s - 1$, the jobs in the positions $i + k$ and $j + k$ are swapped (see Figure 8.6).

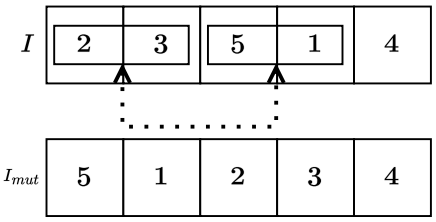

Figure 8.6: Arbitrary batch change with $i = 1$, $j = 3$ and $s = 2$.

8.7 Local Search

The aim of the local search, that we immediately apply after the mutation, is to improve the fitness value of a new offspring and to possibly avoid a premature convergence.

In a lot of scenarios considered in our numerical experiments (see Chapter 9), the local search has turned out to be very effective, since, starting from a good solution produced in the previous phases, it has often provided a perfect weighted balancing between J_A and J_B, i.e. a global solution located in the current neighborhood.

We have adopted the same local search strategy detailed in Algorithm 5 of Chapter 7, proposed for the Lagrangian relaxation approach.

8.8 The Overall Algorithm

The overall genetic algorithm, proposed for solving BAWCT-2, is named GeneticBAWCT-2 and it is detailed in Algorithm 7, where the subroutine $rand([0, 1])$ returns a random real number in the interval [0,1]. The input parameters are the following:

- $size$: size of the population;
- pr_c: probability of performing the crossover;
- pr_m: probability of performing the mutation;
- GenerateIndividual: adopted strategy for generating the initial population;

- Selection: adopted strategy for the selection operator;
- Crossover: adopted strategy for the crossover operator;
- Mutation: adopted strategy for the mutation operator;
- $iter_{max}$: maximum number of iterations.

Algorithm 7: GeneticBAWCT-2

Input: $size$, pr_c, pr_m, $iter_{max}$

$Pop \leftarrow \emptyset$
for $i \leftarrow 1$ ***to*** p_{size} **do**
  $Pop_i \leftarrow$ GenerateIndividual()
  Evaluate(Pop_i)
  $Pop \leftarrow Pop \cup \{Pop_i\}$
$I_{best} \leftarrow$ Best(Pop)
iter $\leftarrow 0$
while $fitness_{I_{best}} < +\infty$ *and* $iter < iter_{max}$ **do**
  $NewPop \leftarrow \emptyset$
  for $i \leftarrow 1$ ***to*** $size$ **do**
    $parents \leftarrow$ Selection(Pop)
    if $rand([0,1]) < pr_c$ **then**
      $offspring \leftarrow$ Crossover($parents$)
    else
      $offspring \leftarrow parents_1$
    if $rand([0,1]) < pr_m$ **then**
      $offspring \leftarrow$ Mutation($offspring$)
    Evaluate($offspring$)
    $offspring \leftarrow$ LocalSearch($offspring$)
    $NewPop \leftarrow NewPop \cup \{offspring\}$
  $toDiscard \leftarrow O \in NewPop$ s.t. $\exists P \in NewPop : fitness_P \geq fitness_O$
  $Pop \leftarrow NewPop \cup \{I_{best}\} \setminus \{toDiscard\}$
  $I_{best} \leftarrow$ Best(Pop)
  iter $\leftarrow$ iter $+1$

Some comments on Algorithm 7 are in order.

Given the current population (Pop), initialized at lines 1-5, at each iteration we create a new population ($NewPop$), by randomly selecting $2size$ numbers in the interval $[0, 1]$ to decide whether, generating the new current individual, crossover and mutation have to be applied. In case the crossover is not carried out, the offspring coincides with the first parent, selected at line 11.

Once an offspring is generated, we proceed with the eventual mutation (line 17, depending on pr_m) and we evaluate the corresponding fitness value (line 18), in order to perform the local search (line 19). Then the new current population (line 22) is obtained by substituting, in the new population, one individual (different from the best one) with the best one of the old population. In this way, we guarantee the monotonicity of the fitness function in corresponding to the best individual (I_{best}) of the current population, which, at the end of the algorithm, constitutes a heuristic solution to the problem.

Finally, we observe that the algorithm terminates either for reaching the maximum number of iterations or because a perfect balancing (i.e. $f(I_{best}) = 0$) has been determined, the latter corresponding to $fitness_{I_{best}} = +\infty$.

Chapter 9

Numerical Experiments

In this chapter, we describe some numerical experiments performed to evaluate the performances of the Lagrangian relaxation approach, presented in Chapter 7, and of the genetic algorithm described in Chapter 8. For the experimentation, we have randomly generated more than one thousand test problems, characterized by a number of jobs up to 2000.

9.1 Introduction

Algorithm 4 (LagrangianBAWCT-2) of Chapter 7 and Algorithm 7 (GeneticBAWCT-2) of Chapter 8 have been implemented in Java (version 14.02) and tested on a Windows 10 system, characterized by 16 GB of RAM and a 2.30 GHz Intel Core i7 processor. For each run, we have fixed a time limit equal to 3600 seconds and the maximum number $iter_{max}$ of iterations equal to 1000. Both the codes stop also when an incumbent equal to zero is generated, which, due to the presence of the absolute value in the objective function of BAWCT-2, corresponds to an optimal solution of the problem providing a perfect weighted balancing between J_A and J_B.

9.2 The Lagrangian Approach versus Gurobi Optimizer

9.2.1 Parameters Setting

As for the initial setting of λ in Algorithm 4 (LagrangianBAWCT-2), we have chosen $\lambda = e$, where e is the vector of ones. To solve the linear assignment problem $LR_x(\lambda)$, we have used Gurobi Optimizer software, version 9.1. Regarding the choice of the initial sequence $\bar{\pi}_{init}$, we have set it as the sequence obtained by alternating the jobs between the two classes until one class is completely scheduled, and by accommodating at the end the remaining jobs of the other class, if any.

We observe that, in the mixed integer linear model (5.3)-(5.12), the parameter b_t provided by the Glover linearization can be interpreted as a sort of big M, since it constitutes an upper bound for the variable z_{jt}. Based on this consideration, in all the numerical experiments we have used a more stringent value, say $\bar{b}_t$, for this parameter, letting

$$\bar{b}_1 \stackrel{\triangle}{=} 0$$

and

$$\bar{b}_t \stackrel{\triangle}{=} \sum_{l=1}^{t-1} p_{(l)}, \quad t = 2, \ldots, n,$$

with $p_{(j)}$, for $j = 1, \ldots, n$, being the processing times such that

$$p_{(1)} \geq p_{(2)} \geq \cdots \geq p_{(n)}.$$

This choice does not affect the equivalence between problem (5.3)-(5.12) and problem (5.2) (see Theorem 5.2.1), since, taking into account the assignment constraints (5.9)-(5.12), it holds

$$\sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \leq \bar{b}_t, \quad t = 1, \ldots, n.$$

9.2.2 Numerical Results

The LagrangianBAWCT-2 code has been first tested on 21 test problems listed in Table 9.1 and obtained for different values of n_A and n_B, up to 500 jobs.

The processing times and the weights have been randomly generated from a uniform distribution on the interval $[1, 25]$ for problems 1-12 (small instances), on the interval $[1, 50]$ for problems 13-16 (medium instances) and on the interval $[1, 100]$ for problems 17-21 (large instances).

We compare our results with those ones obtained by Gurobi Optimizer in solving exactly the mixed integer quadratically constrained program (5.2) and the mixed integer linear program (5.3)-(5.12) (columns MIQCP and MILP of Table 9.1, respectively), for which we have fixed as well a time limit equal to 3600 seconds. Moreover, in order to force Gurobi to generate better feasible solutions, we have set the parameters MIPFocus and Heuristics equal to 1.

#	n	n_A	n_B	Greedy	Lagrangian heuristics		MIQCP (Gurobi)		MILP (Gurobi)	
				Incumbent	Time (secs)	Incumbent	Time (secs)	Incumbent	Time (secs)	Incumbent
1	20	10	10	525.50	0.021	0	139.73	0	0.940	0
2	30		20	1194.75	0.017	0	3600	0.50	3.05	0
3	40		30	1446.23	0.031	0	3600	2805.10	7.55	0
4	40	20	20	836.65	0.019	0	3600	0.350	14.20	0
5	50		30	2156.72	0.031	0	3600	6227.03	42.58	0
6	60		40	2049.95	0.123	0	3600	-	41.85	0
7	60	30	30	585.77	0.049	0	3600	-	61.46	0
8	70		40	2488.88	0.069	0	3600	-	128.76	0
9	80		50	1537.95	0.242	0	OOM		361.27	0
10	80	40	40	279.83	0.337	0	OOM		145.09	0
11	90		50	342.63	1.69	0	OOM		250.51	0
12	100		60	2922.91	0.288	0	OOM		528.43	0
13	100	50	50	5203.38	0.067	0	OOM		532.79	0
14	150		100	31312.22	1.02	0	OOM		3600	376.26
15	200	100	100	4066.43	3.26	0	OOM		OOM	
16	250		150	25109.42	1.49	0	OOM		OOM	
17	300	150	150	5520.81	2.97	0	OOM		OOM	
18	350		200	82117.30	97.47	0	OOM		OOM	
19	400	200	200	67275.48	146.49	0	OOM		OOM	
20	450		250	172938.22	41.95	0	OOM		OOM	
21	500	250	250	39367.53	128.12	0	OOM		OOM	

Table 9.1: Comparison of LagrangianBAWCT-2 against Gurobi Optimizer

In particular, for each solver we report the execution time and the best objective function value found by the algorithm (the incumbent). The character "-" means that no feasible solution has been found within the time limit, while "OOM" indicates an out-of-memory failure.

In solving each problem, LagrangianBAWCT-2 has exited with the best incumbent being equal to zero, which is a sign of optimality (perfect weighted balancing between J_A and J_B). We remind, in fact, that zero is as well a trivial lower bound for the original problem because of the absolute value in the objective function.

Note also that our approach is very fast, whereas solving exactly the mixed integer programs appears to be prohibitive from the computational point of view: in the quadratically constrained case (column MIQCP), in most of the instances Gurobi Optimizer is not able to provide a feasible solution or fails because of the out-of-memory event, while in the linear case (column MILP) an optimal solution is found prevalently on the small instances, whereas for the medium and the large ones the out-of-memory event occurs.

A trivial greedy solution to problem (5.1) is that one obtainable by alternatively scheduling the jobs of the two classes and used to initialize $\bar{\pi}_{init}$. Then, in order to get an idea of how far this trivial solution is from the optimal one, we have added in Table 9.1 the values of the corresponding incumbent (column Greedy Incumbent), which clearly testify that the greedy approach provides a very imbalanced solution.

To assess the Lagrangian heuristics results of Table 9.1, we have performed further experiments by testing our code on three additional groups of problems (small, medium and large), considering different scenarios by varying n_A and n_B chosen in the same way as in Table 9.1 (see Tables 9.2-9.4). For each scenario a set of 50 instances has been generated, adopting the same above strategy in order to randomly obtain the processing times and the weights.

In Tables 9.2-9.4, for each scenario we report the following results:

- the average execution time (seconds);
- the average value of the best incumbent;
- the average number of iterations needed to compute the best incumbent;
- the average execution time (seconds) per iteration.
- the number of instances certainly solved to optimality, i.e. when the generated incumbent is equal to zero (in the other cases the optimal

n	n_A	n_B	Time (secs)	Incumbent	# Iter	Time per iter (secs)	# Solved to opt
20		10	0.041	0	16.38	0.003	50/50
30	10	20	0.039	0	7.26	0.005	50/50
40		30	0.041	0	4.44	0.009	50/50
40		20	0.062	0	6.52	0.010	50/50
50	20	30	0.158	0	9.74	0.016	50/50
60		40	0.092	0	4.01	0.023	50/50
60		30	0.079	0	3.50	0.023	50/50
70	30	40	0.248	0	7.18	0.035	50/50
80		50	0.346	0	7.08	0.049	50/50
80		40	0.122	0	2.32	0.053	50/50
90	40	50	0.651	0	8.96	0.073	50/50
100		60	0.261	0	2.96	0.088	50/50

Table 9.2: LagrangianBAWCT-2 on small test problems ($p_j, w_j \in [1, 25]$)

objective function value is unknown and we are not able to verify the optimality).

The results reported in Tables 9.2-9.4 are coherent with those ones reported in Table 9.1, since in solving each problem the code has exited very fast with the best incumbent being again equal to zero.

To make the numerical experiments more compelling, we have run the code also on some additional instances characterized by larger values of p_j and w_j, which have been taken randomly as integer numbers from a uniform distribution on the interval $[1, 3n]$ (see Table 9.5). In such case, we have considered six scenarios and for each scenario we have generated 20 instances. Differently from the previous type of experiments, the code did not exit always with the best incumbent equal to zero, within the fixed time limit. Nevertheless, on the other hand, it is possible to observe that in all the cases an incumbent less than or equal to 0.005 has been found on average.

n	n_A	n_B	Time (secs)	Incumbent	# Iter	Time per iter (secs)	# Solved to opt
100	50	50	0.514	0	5.54	0.093	50/50
150		100	1.04	0	3.48	0.300	50/50
200	100	100	1.89	0	2.54	0.744	50/50
250		150	8.35	0	4.82	1.73	50/50

Table 9.3: LagrangianBAWCT-2 on medium test problems ($p_j, w_j \in [1, 50]$)

n	n_A	n_B	Time (secs)	Incumbent	# Iter	Time per iter (secs)	# Solved to opt
300	150	150	20.99	0	5.14	4.08	50/50
350		200	65.91	0	10.50	6.28	50/50
400	200	200	66.10	0	3.38	19.56	50/50
450		250	212.55	0	8.56	24.83	50/50
500	250	250	123.29	0	3.64	33.87	50/50

Table 9.4: LagrangianBAWCT-2 on large test problems ($p_j, w_j \in [1, 100]$)

From the overall results reported in Tables 9.2-9.5, it is evident how the average time per iteration is monotonically increasing with respect to the number of jobs. This behaviour is not surprising, since solving the assignment problem and performing the local search require clearly more computational effort for large values of n.

It is worth noting that, even if having $C^* = 0$ is very frequent in the practical cases (as shown by our randomly generated instances), on the other hand this is not always guaranteed. Then, for the sake of completeness, in order to analyze the behaviour of the Lagrangian heuristics in solving a non-perfectly balanced problem (i.e. a problem with $C^* > 0$), we have constructed a simple toy example characterized by 10 jobs and $C^* = 17$, whose data are

n	n_A	n_B	Time (secs)	Incumbent	# Iter	Time per iter (secs)	# Solved to opt
60	30	30	1.08	0	95.45	0.011	20/20
100	50	50	12.01	0	124.10	0.097	20/20
200	100	100	144.58	0	178.10	0.812	20/20
300	150	150	737.13	0	177.00	4.17	20/20
400	200	200	1467.99	0.001	138.05	10.63	16/20
500	250	250	1847.42	0.005	81.70	22.61	4/20

Table 9.5: LagrangianBAWCT-2 on test problems with large ranges ($p_j, w_j \in [1, 3n]$)

listed in Table 9.6.

J_A		J_B	
p_j	w_j	p_j	w_j
5	5	7	10
3	3	3	11
1	4	9	16
6	2	2	15
5	1	4	9

Table 9.6: Toy problem with $n_A = n_B = 5$ and $C^* = 17$

The results obtained by the Lagrangian heuristics on the toy problem are reported in Table 9.7, in comparison with those ones provided by Gurobi Optimizer in solving the two mixed integer problems, the quadratically constrained program (MICQP) and the linear one (MILP). LagrangianBAWCT-2 has exited for reaching the maximum number $iter_{max}$ of iterations, performed in 0.758 seconds. On the other hand, Gurobi Optimizer has solved the quadratically constrained program in 91.24 seconds, while in the linear case it has determined the optimal solution in 20.15 seconds.

To confirm the results obtained on the above toy problem, we finally report in Table 9.8 the results obtained by the Lagrangian heuristics and Gurobi Optimizer on additional 20 test problems, characterized by unweighted iden-

Lagrangian heuristics		MIQCP (Gurobi)		MILP (Gurobi)	
Time (secs)	Incumbent	Time (secs)	Incumbent	Time (secs)	Incumbent
0.758	17.00	91.24	17.00	20.15	17.000

Table 9.7: Comparison LagrangianBAWCT-2 against Gurobi Optimizer on the toy problem described in Table 9.6

tical jobs (i.e. with $p_j = p$ and $w_j = 1$, for all $j \in J$) and n_A and n_B odd numbers. These problems, for which the optimal objective function value is computable a priori as $C^* = \frac{np}{2n_A n_B} > 0$ (see Corollary 6.3.7 of Chapter 6), have been obtained by randomly generating the constant p (fifth column of Table 9.8) from a uniform distribution on the same intervals used for generating the instances of Table 9.1, i.e. $[1, 25]$ for problems 1-12 (small instances), $[1, 50]$ for problems 13-16 (medium instances) and $[1, 100]$ for problems 17-21 (large instances). In all such cases, the Lagrangian heuristics code has been able to compute an optimal solution, exiting either for the maximum number of iterations or for reaching the time limit.

About the comparison against Gurobi, we can confirm the comments done on the results of Table 1, even if, in solving the MILP model, Gurobi is faster than the Lagrangian heuristics on some of the small instances. Also for such kind of problems, the greedy approach, for which in these cases the objective function value is easily computable in a closed form[1], provides an objective function value that is always far from the optimal one.

[1] $\left|\bar{C_B} - \bar{C_A}\right| = \frac{p\left(\max\{n_A, n_B\}^2 - \min\{n_A, n_B\}^2 + n_A + n_B\right)}{2\max\{n_A, n_B\}}$

#	n	n_A	n_B	p	Greedy	Lagrangian heuristics		MIQCP (Gurobi)		MILP (Gurobi)	
					Incumbent	Time (secs)	Incumbent	Time (secs)	Incumbent	Time (secs)	Incumbent
1	30		15	11	11.00	3.05	0.733	3600	0.733	0.68	0.733
2	40	15	25	9	79.20	8.16	0.480	3600	180.00	2.22	0.480
3	50		35	10	150.00	9.17	0.476	3600	250.00	5.71	0.476
4	50		25	13	13.00	10.45	0.520	3600	325.00	5.81	0.520
5	60	25	35	13	122.57	19.75	0.446	3600	-	14.20	0.446
6	70		45	12	196.00	42.31	0.373	OOM		29.88	0.373
7	70		35	12	12.00	21.15	0.343	OOM		3600	0.343
8	80	35	45	15	146.67	31.47	0.381	OOM		56.28	0.381
9	90		55	15	257.73	44.87	0.351	OOM		102.37	0.351
10	90		45	14	14.00	43.33	0.311	OOM		102.76	0.311
11	100	45	55	14	140.00	60.28	0.283	OOM		3600	0.283
12	110		65	17	302.08	76.23	0.319	OOM		3600	0.319
13	150	75	75	43	43.00	211.41	0.573	OOM		3600	517.72
14	200		125	16	652.80	733.44	0.171	OOM		OOM	
15	250	125	125	42	42.00	1927.55	0.336	OOM		OOM	
16	300		175	2	87.43	3378.99	0.014	OOM		OOM	
17	350	175	175	77	77.00	3600	0.440	OOM		OOM	
18	400		225	1	45.33	3600	0.005	OOM		OOM	
19	450	225	225	26	26.00	3600	0.116	OOM		OOM	
20	500		275	54	2503.64	3600	0.218	OOM		OOM	

Table 9.8: Comparison of LagrangianBAWCT-2 against Gurobi Optimizer on test problems with unweighted identical jobs and both n_A and n_B odd

9.3 The Genetic Algorithm versus the Lagrangian Approach

9.3.1 Parameters Setting

We recall that a preliminary step in the implementation of any genetic algorithm is the parameters setting, which is crucial since it strongly affects the performance of the approach. Moreover, because a vast number of possible settings has to be considered, this is not a trivial step.

In particular, our heuristics GeneticBAWCT-2 (Algorithm 7) is defined in

function of three numerical input parameters ($size$, pr_c, pr_m) and four categorical parameters (the inner procedures *Initial population*, *Selection*, *Crossover* and *Mutation*), the latter to be chosen among the different strategies proposed in the previous chapter and summarized in Table 9.9. About the possible values of the numerical parameters, our proposal is reported in Table 9.10 (three different values for each parameter), coming out with a total of seven parameters and $3^7 = 2187$ combinations for the overall setting of the algorithm. Due to the huge number of these combinations, the parameters setting has been simplified by adopting the following strategy.

Parameters	Proposed values		
Initial population	Random	Alternated	Bidirectional
Selection	Roulette	Binary tournament	k-tournament
Crossover	One-point	Two-point	Position-based
Mutation	Arbitrary two-job	Shift	Arbitrary batch

Table 9.9: Domains of the categorical parameters

.

Parameters	Proposed values		
$size$	20	50	100
pr_c	0.75	0.85	0.95
pr_m	0.05	0.25	0.5

Table 9.10: Domains of the numerical parameters

Initially, the three numerical parameters ($size$, pr_c, pr_m) have been fixed to the median of the proposed values, i.e. 50, 0.85 and 0.25, respectively. Then, in correspondence to such values of the numerical parameters, we have considered all the 3^4 combinations of the four categorical parameters and, for each combination, we have run the code on 30 randomly generated instances of the problem. For creating such instances, we have considered three different scenarios in correspondence to $n_A = n_B = k$, with $k \in \{100, 150, 250\}$, and, for each scenario, we have generated ten instances by uniformly sampling p_j and w_j in the interval $[1, 3n]$. Taking into account that the algorithm has

been able to solve to optimality (i.e. $fitness_{I_{best}} = +\infty$) all these instances, the execution time has been considered as the performance measure. The optimal setting of the categorical parameters, i.e. that one in correspondence to which we have obtained the lowest average execution time, has resulted in the following:

- Initial population: Bidirectional generation;
- Selection: Binary tournament;
- Crossover: Two-point crossover;
- Mutation: Shift change.

Successively, in correspondence to the above fixed best configuration of the categorical parameters, we have similarly proceeded to determine the optimal setting of the numerical parameters by considering all the 3^3 combinations. We have obtained:

- $size = 20$;
- $pr_c = 0.85$;
- $p_m = 0.5$.

For the sake of completeness, in Figures 9.1 and 9.2, the main effects plots of the categorical and of the numerical parameters, respectively, are reported, using as metric the execution time.

9.3.2 Numerical Results

The main purpose of this section is to show the effectiveness of the genetic algorithm especially in solving very large scale problems. Then, in order to compare GeneticBAWCT-2 against LagrangianBAWCT-2, we have run the codes on the same large test problems listed in Table 9.5, reporting the results in Table 9.11. In particular, for each scenario we list:

- the average execution time (in seconds);
- the average best incumbent, i.e. the average best objective function value of problem (4.1), found by the algorithm;

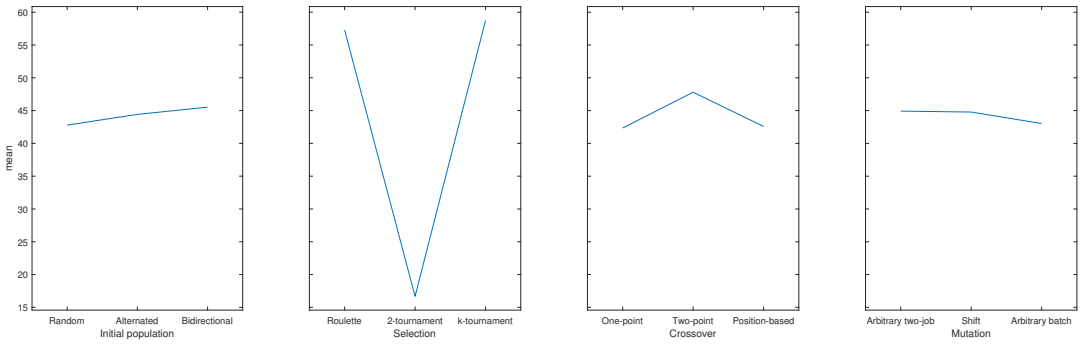

Figure 9.1: Main effects plots of the categorical parameters

- the number of instances certainly solved to optimality, i.e. how many times the algorithm exited with a zero objective function value.

			LagrangianBAWCT-2			GeneticBAWCT-2		
n	n_A	n_B	Time (secs)	Best incumbent	#Solved to opt	Time (secs)	Best incumbent	#Solved to opt
60	30	30	1.08	0	20/20	0.026	0	20/20
100	50	50	12.01	0	20/20	0.101	0	20/20
200	100	100	144.58	0	20/20	1.35	0	20/20
300	150	150	737.13	0	20/20	3.45	0	20/20
400	200	200	1467.99	0.001	16/20	11.97	0	20/20
500	250	250	1847.42	0.005	4/20	42.72	0	20/20

Table 9.11: Comparison of GeneticBAWCT-2 against LagrangianBAWCT-2

Looking at the results, it is evident that the genetic algorithm overcomes the Lagrangian heuristics: in fact, differently from LagrangianBAWCT-2, the GeneticBAWCT-2 code has been able to solve to optimality all the 120 instances, and moreover, in terms of execution time, it is clearly the winner with a difference of two orders of magnitude.

In order to test this approach on more and larger instances, we have in addition randomly generated new large scale test problems grouped in seven scenarios, each of them constituted by ten instances. In such case, a maximum number of jobs equal to 2000 has been considered, taking again p_j and w_j as

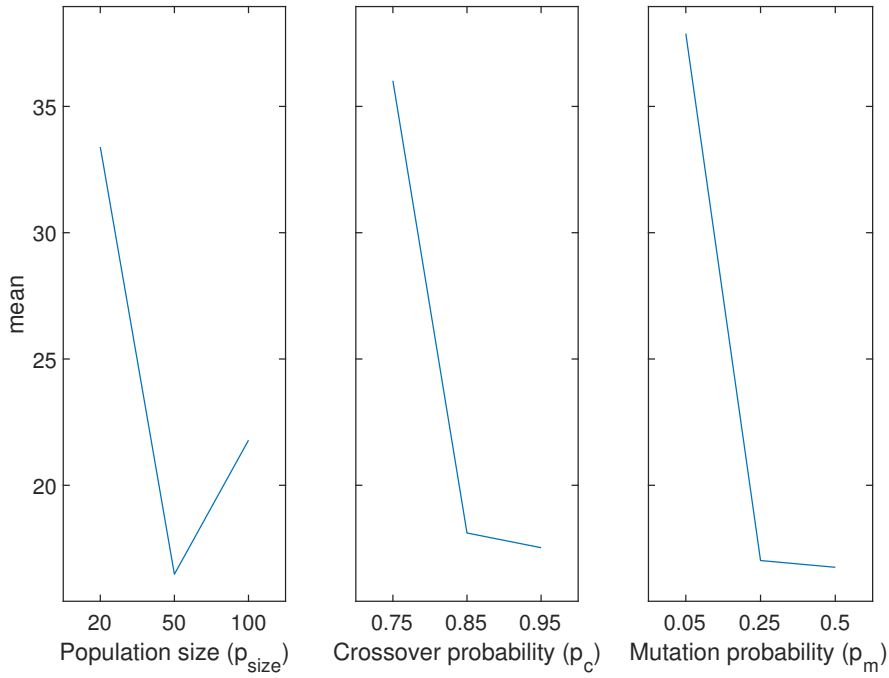

Figure 9.2: Main effects plots of the numerical parameters

random integer numbers from a uniform distribution on the interval $[1, 3n]$. The obtained results are reported in Table 9.12, from which we can observe that the genetic algorithm GeneticBAWCT-2 has been able to certainly solve to optimality more than 50% of the overall instances, providing however, for each scenario, an average value of the incumbent less than 0.001.

n	n_A	n_B (secs)	Time incumbent	Best to opt	#Solved
1000	500	500	619.81	0	10/10
1100	500	600	2581.38	0.0002	5/10
1200	500	700	2370.27	0.0001	5/10
1300	500	800	2589.24	0.0002	4/10
1400	500	900	3220.57	0.0002	4/10
1500	500	1000	1984.54	0.0002	8/10
2000	1000	1000	2426.04	0.0006	6/10

Table 9.12: LagrangianBAWCT-2 on very large test problems ($p_j, w_j \in [1, 3n]$)

Part III

Conclusions

Chapter 10

Conclusions and Future Research

In this thesis, we have introduced a new single-machine scheduling problem for balancing the average weighted completion times of two classes of jobs. This problem, named BAWCT-2, could be interpreted as a two-agent scheduling problem of the cooperative type. In fact, even if the agents share the same machine, they cooperate in order to optimize the (unique) global objective function, aimed at balancing their average weighted completion times.

After discussing some possible applications, we have proved the NP-hardness of the problem by a polynomial reduction from the well known NP-complete partition problem.

Then, focusing on a simplified version of the problem obtained by assuming identical jobs (i.e. same processing time) and unitary weight, we have shown that the problem in such case is polynomially solvable. In particular, we have proposed a resolution algorithm, named OptBACTI-2, and we have proved that it outputs an optimal solution in linear time.

Successively, in order to face the general NP-hard problem, we have proposed a possible mathematical formulation, obtaining a variant of the well known quadratic assignment problem that can be easily reduced to a mixed integer quadratically constrained formulation. Then, by applying the Glover linearization technique, we have obtained a mixed integer linear program, exploited to design a Lagrangian heuristics based on solving, at each iteration, a linear assignment problem. The proposed algorithm, named Lagrangian-BAWCT-2, has revealed to be able to solve instances up to 500 jobs. As a

consequence, in order to face larger scale instances, we have proposed a genetic algorithm to speed up the resolution process. The algorithm, named GeneticBAWCT-2, has been able to deal with instances containing up to 2000 jobs.

Future research will be devoted to consider a bi-objective version of the problem, introducing a new criterion aimed at minimizing the average weighted completion time of one of the two agents. In fact, in the current version of the problem faced in the thesis, the possibility of obtaining balanced solutions with high values of the average weighted completion times of the two agents can occur.

Very recently, we have also modelled BAWCT-2 as a network design problem. Such formulation could be exploited to design new potential heuristic solution approaches.

Part IV

References

References

[1] E. Aarts and J.K. Lenstra. *Local search in combinatorial optimization.* John Wiley & Sons, Inc., USA, 1997.

[2] L. R. Abreu, J. O. Cunha, B. A. Prata, and J. M. Framinan. A genetic algorithm for scheduling open shops with sequence-dependent setup times. *Computers and Operations Research*, 113, 2020.

[3] A. Agnetis, J.-C. Billaut, S. Gawiejnowicz, D. Pacciarelli, and A. Soukhal. *Multiagent scheduling: Models and algorithms.* Springer, 2014.

[4] A. Agnetis, J.-C. Billaut, S. Gawiejnowicz, D. Pacciarelli, and A. Soukhal. *Multiagent scheduling: Models and algorithms.* Springer, 2014.

[5] A. Agnetis, B. Chen, G. Nicosia, and A. Pacifici. Price of fairness in two-agent single-machine scheduling problems. *European Journal of Operational Research*, 276(1):79–87, 2019.

[6] A. Agnetis, G. De Pascale, and D. Pacciarelli. A lagrangian approach to single-machine scheduling problems with two competing agents. *Journal of Scheduling*, 12(4):401–415, 2009.

[7] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2):229–242, 2004.

[8] A. Agnetis, D. Pacciarelli, and A. Pacifici. Multi-agent single machine scheduling. *Annals of Operations Research*, 150(1):3–15, 2007.

[9] M. Avolio and A. Fuduli. A Lagrangian heuristics for balancing the average weighted completion times of two classes of jobs in a single-machine

scheduling problem. *EURO Journal on Computational Optimization*, 10:100032, 2022.

[10] E. Bahel and C. Trudeau. Stability and fairness in the job scheduling problem. *Games and Economic Behavior*, 117:1–14, 2019.

[11] K. R. Baker and J. C. Smith. A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6(1):7–16, 2003.

[12] D. Bertsimas, V. F. Farias, and N. Trichakis. The price of fairness. *Operations research*, 59(1):17–31, 2011.

[13] S.J. Brams and A. D. Taylor. *Fair division - from cake-cutting to dispute resolution*. 1996.

[14] R.E. Burkard, E. Çela, P.M. Pardalos, and L.S. Pitsoulis. The quadratic assignment problem. In D.Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 241–337. Springer US, Boston, MA, 1998.

[15] M. Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[16] J. Fan, C. Zhang, Q. Liu, W. Shen, and L. Gao. An improved genetic algorithm for flexible job shop scheduling problem considering reconfigurable machine tools with limited auxiliary modules. *Journal of Manufacturing Systems*, 62:650–667, 2022.

[17] A. Frangioni. About Lagrangian methods in integer optimization. *Annals of Operations Research*, 139(1):163–193, 2005.

[18] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[19] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[20] M. Gaudioso. *A view of Lagrangian relaxation and its applications*, pages 579–617. Numerical Nonsmooth Optimization: State of the Art Algorithms. 2020.

[21] M. Gaudioso, G. Giallombardo, and G. Miglionico. Essentials of numerical nonsmooth optimization. *4OR*, 18(1):1–47, 2020.

[22] A. M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22(3):618–630, 1968.

[23] F. Glover. Improved linear integer programming formulations of nonlinear integer problems. *Management Science*, 22(4):455–460, 1975.

[24] J. F. Gonçalves, J. J. De Magalhães Mendes, and M. G. C. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167(1):77–95, 2005.

[25] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. *Optimization and approximation in deterministic sequencing and scheduling: A survey*, volume 5 of *Annals of Discrete Mathematics*. 1979.

[26] M. Guignard. Lagrangean relaxation. *Top*, 11(2):151–200, 2003.

[27] J. R. Jackson. Scheduling a production line to minimize maximum tardiness. 1955.

[28] L. Kaufman and F. Broeckx. An algorithm for the quadratic assignment problem using Bender's decomposition. *European Journal of Operational Research*, 2(3):207–211, 1978.

[29] B. Kazimipour, X. Li, and A. K. Qin. A review of population initialization techniques for evolutionary algorithms. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, pages 2585–2592, 2014.

[30] O. Kramer. *Genetic Algorithm Essentials*, volume 679 of *Studies in Computational Intelligence*. Springer, 2017.

[31] P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170, 1999.

[32] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of Machine Scheduling Problems. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.

[33] S.-S. Li, R.-X. Chen, and J. Tian. Multitasking scheduling problems with two competitive agents. *Engineering Optimization*, 52(11):1940–1956, 2020.

[34] S.-S. Li and J.-J. Yuan. Single-machine scheduling with multi-agents to minimize total weighted late work. *Journal of Scheduling*, 23(4):497–512, 2020.

[35] P. Liu, M. Gu, and G. Li. Two-agent scheduling on a single machine with release dates. *Computers and Operations Research*, 111:35–42, 2019.

[36] Z. Liu, J. Wang, C. Zhang, H. Chu, G. Ding, and L. Zhang. A hybrid genetic-particle swarm algorithm based on multilevel neighbourhood structure for flexible job shop scheduling problem. *Computers and Operations Research*, 135, 2021.

[37] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[38] B. Moseley and S. Vardi. The efficiency-fairness balance of Round Robin scheduling. *Operations Research Letters*, 50(1):20–27, 2022.

[39] Q.Q. Nong, T.C.E. Cheng, and C.T. Ng. Two-agent scheduling to minimize the total cost. *European Journal of Operational Research*, 215(1):39 – 44, 2011.

[40] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers and Operations Research*, 35(10):3202–3212, 2008.

[41] A. Phu-ang and A. Thammano. Memetic algorithm based on marriage in honey bees optimization for flexible job shop scheduling problem. *Memetic Computing*, 9(4):295–309, 2017.

[42] M. L. Pinedo. *Scheduling: Theory, algorithms, and systems, fifth edition*. Springer International Publishing, 2016.

[43] B.T. Polyak. *Introduction to optimization*. Optimization Software Inc., New York, 1987.

[44] P. W. Poon and J. N. Carter. Genetic algorithm crossover operators for ordering applications. *Computers and Operations Research*, 22(1):135–147, 1995.

[45] J. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:339–370, 1996.

[46] M. R. Raeesi N. and Z. Kobti. A memetic algorithm for job shop scheduling using a critical-path-based local search heuristic. *Memetic Computing*, 4(3):231–245, 2012.

[47] N.Z. Shor. *Minimizations methods for nondifferentiable functions.* Springer-Verlag, Berlin, 1985.

[48] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[49] M. Tan, H. . Yang, and Y. . Su. Genetic algorithms with greedy strategy for green batch scheduling on non-identical parallel machines. *Memetic Computing*, 11(4):439–452, 2019.

[50] A. Umbarkar and P. D. Sheth. Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6(1), 2015.

[51] X. Wang, T. Ren, D. Bai, C. Ezeh, H. Zhang, and Z. Dong. Minimizing the sum of makespan on multi-agent single-machine scheduling with release dates. *Swarm and Evolutionary Computation*, 69:100996, 2022.

[52] S. L. Yadav and A. Sohal. Comparative study of different selection techniques in genetic algorithm. *International Journal of Engineering, Science and Mathematics*, 6(3):174–180, 2017.

[53] Y. Yin, W. . Wu, S. . Cheng, and C. . Wu. An investigation on a two-agent single-machine scheduling problem with unequal release dates. *Computers and Operations Research*, 39(12):3062–3073, 2012.

[54] C. Yu, Q. Semeraro, and A. Matta. A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility. *Computers and Operations Research*, 100:211–229, 2018.

[55] J. Yuan, C.T. Ng, and T.C.E. Cheng. Scheduling with release dates and preemption to minimize multiple max-form objective functions. *European Journal of Operational Research*, 280(3):860–875, 2020.

[56] G. Zhang, Y. Hu, J. Sun, and W. Zhang. An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm and Evolutionary Computation*, 54, 2020.

[57] Y. Zhang, Z. Zhang, and Z. Liu. The price of fairness for a two-agent scheduling game minimizing total completion time. *Journal of Combinatorial Optimization*, in press, 2020.

Part V

Publications

Information Processing Letters 155 (2020) 105886

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A subset-sum type formulation of a two-agent single-machine scheduling problem

Matteo Avolio, Antonio Fuduli *

Department of Mathematics and Computer Science, University of Calabria, Rende, Italy

ARTICLE INFO

Article history:
Received 24 July 2019
Received in revised form 23 October 2019
Accepted 1 November 2019
Available online 5 November 2019
Communicated by K. Buchin

Keywords:
Scheduling
Single-machine
Completion time
Subset-sum

ABSTRACT

We tackle a single-machine scheduling problem, characterized by two sets of identical jobs contributing to a common objective function, aimed at balancing the average completion times.
For solving this problem, that we show it reduces to a particular case of the subset-sum problem, we propose an exact algorithm, pointing out the existence of different optimal solutions. Without considering explicitly the job-position assignment procedure, which necessarily would require at least a linear time, the problem is solvable in constant time. A variant of such a problem is also considered.

1. Introduction

There exists a huge variety of scheduling problems [1], depending on the number and/or on the type of machines, on the characteristics of the jobs and on the prefixed objective function. Most of them are characterized by one decision-maker (the *agent*) and one objective function. On the other hand, in the last years some researchers have focused on the so-called *multi-agent* scheduling problems, where each agent has to process its jobs on some machines to be shared with other agents. We highlight that a multi-agent scheduling problem is different from a multi-objective one: in the multi-objective case (see for example [2]), all the jobs contribute to each objective function, whereas, in the multi-agent case, generally each agent submits its jobs according to its own objective function, which may be different and/or opposite with respect to the objective functions of its competitors. In other words, in a multi-agent problem each job generally contributes only to the objective of the corresponding agent.

One of the seminal papers on two-agent single-machine scheduling problem is [3], where different scenarios has been considered by taking into account the following objective functions: the maximum of regular functions, the number of late jobs and the total weighted completion times. Other papers on two-agent single-machine scheduling problems are for example [4–7].

An interesting multi-agent problem is given in [8], where, apart from the objective functions relative to each subset of jobs, a global objective function is at the same time considered, taking into account the global performance of the overall system. Finally, an extension to multiple machines has been presented in [9].

As mentioned above, in a standard multi-agent scheduling problem each job contributes only to the objective of the corresponding agent: for this reason we say that this kind of problem is of the *competitive* type.

In contrast, in this paper we tackle a two-agent single-machine *co-operative* type scheduling problem, non-preemptive and deterministic, characterized by two sets of jobs, one for each agent. All the involved jobs contribute to a common objective function, aimed at balancing the

* Corresponding author.
E-mail addresses: matteoavolio0@gmail.com (M. Avolio), antonio.fuduli@unical.it (A. Fuduli).

https://doi.org/10.1016/j.ipl.2019.105886
0020-0190/

average completion times between the two sets. For this kind of problem we present a solution algorithm under the hypothesis that all the jobs in the two sets are identical (i.e. they have the same processing time), showing that an optimal solution is obtainable in constant time if the job-position assignment procedure is not explicitly considered.

We point out that, although this problem reduces to a simple case of the well known subset-sum problem, to the best of our knowledge it seems that it has not been expressly treated in the literature.

The paper is organized as follows. In the next section we formalize the problem, showing that it reduces to a particular case of the subset-sum problem. In Section 3 we describe a solution algorithm, pointing out the existence of different optimal solutions and providing a possible closed-form expression for solving the problem. Finally the optimality is proved in Section 4 and a variant of such a problem is considered in Section 5.

Throughout the paper, we adopt the following notation. Given a sequence π and a job j, we indicate by $[j]$ the integer number indicating the position of job j in the sequence π and by $\cdot_{[j]}$ any quantity referred to the job processed in the position j. Finally, in correspondence to any real number r, the nearest integer number less (resp. greater) than or equal to r is denoted by $\lfloor r \rfloor$ (resp. $\lceil r \rceil$).

2. Problem definition

In this section we formalize the proposed two-agent single-machine scheduling problem, assuming that all the involved jobs are identical.

Consider the following example. Two groups of students, A and B, coming from two different courses, have to take an oral examination with the same teacher. The teacher convokes all of them at the same hour in the lecture room and he would like to *schedule* them in a such way that the average completion times of both the groups are possibly the same. This problem could be modeled as follows.

A and B are the agents and the students of each group are the jobs of the agents. Then we indicate by

$$J_A \triangleq \{a_1, \ldots, a_{n_A}\}$$

the job set of the agent A and by

$$J_B \triangleq \{b_1, \ldots, b_{n_B}\}$$

the job set of the agent B, with $n_A, n_B \geq 1$. We assume that all the jobs in the sets J_A and J_B are non-preemptive and characterized by the following nonnegative processing times:

$$p_{a_i} \quad i = 1, \ldots, n_A$$

and

$$p_{b_i} \quad i = 1, \ldots, n_B.$$

The goal of both the agents is to process their jobs on a same machine, with the aim to balance as much as possible the respective average completion times. Then, indicating by π any possible schedule of the jobs, the optimization problem to be solved is the following:

$$\min_{\pi} \left| \frac{\sum_{i=1}^{n_A} C_{a_i}(\pi)}{n_A} - \frac{\sum_{i=i}^{n_B} C_{b_i}(\pi)}{n_B} \right|, \tag{1}$$

where $C_{a_i}(\pi)$, $i = 1, \ldots, n_A$, and $C_{b_i}(\pi)$, $i = 1, \ldots, n_B$, are the completion times (depending on the specific sequence π) of the jobs in the sets J_A and J_B, respectively.

Letting $n \triangleq n_A + n_B$, we focus on the simplest case where we assume that all the n jobs have the same processing time, say $p > 0$. Then we come out with the following optimization problem:

$$\begin{cases} C^* \triangleq \min\limits_{\pi} \left| \dfrac{\sum_{i=1}^{n_A} C_{a_i}(\pi)}{n_A} - \dfrac{\sum_{i=i}^{n_B} C_{b_i}(\pi)}{n_B} \right| \\ \text{with:} \\ p_{a_i} = p \quad i = 1, \ldots, n_A \\ p_{b_i} = p \quad i = 1, \ldots, n_B, \end{cases} \tag{2}$$

which can be interpreted as a kind of subset-sum problem [10]. In fact, since all the n jobs have the same processing time p, in correspondence to any sequence π we have:

$$\begin{aligned} &C_{[1]} = p, \\ &C_{[2]} = 2p, \\ &\vdots \\ &C_{[n]} = np, \end{aligned} \tag{3}$$

where $C_{[j]}$ is the completion time of the job processed in the position j (for the sake of simplicity in the notation, we omit the dependence of the completion time on π, if no confusion occurs). Then, in correspondence to any subset $\bar{I} \subseteq I = \{1, \ldots, n\}$, we have

$$\sum_{i \in \bar{I}} C_{[i]} = p \sum_{i \in \bar{I}} i.$$

As a consequence, problem (2) reduces to finding two disjoint subsets $\bar{I}_A$ and $\bar{I}_B$ of I, having cardinality n_A and n_B respectively, such that $\bar{I}_A \cup \bar{I}_B = I$ and the quantity

$$p \left| \frac{\sum_{i \in \bar{I}_A} i}{n_A} - \frac{\sum_{i \in \bar{I}_B} i}{n_B} \right|$$

is minimum.

3. A solution algorithm

In this section we present an algorithm, which provides an optimal solution to problem (2), assigning each job to exactly one position into the sequence.

Given the job set J_A of the agent A and the job set J_B of the agent B, together with their respective cardinalities n_A and n_B, the algorithm works by first assigning to some

appropriate positions of the optimal sequence all the jobs of one set, that in the sequel we will call the *pioneer set*, and by successively accommodating the jobs of the other set into the remaining positions. The choice of the pioneer set is made in function of the parity of n_A and n_B. In particular, when n_A and n_B are both even or both odd, the sets J_A and J_B play a symmetric role and, in this case, we can choose indifferently one of them as the pioneer set. On the other hand, if the cardinalities n_A and n_B of the two sets are neither both odd nor both even, the role played by J_A and J_B is no more symmetric and the pioneer set must necessarily be the set whose cardinality is even.

Based on these considerations, without loss of generality, we state the algorithm assuming J_A as the pioneer set. In fact, whenever n_A is odd and n_B is even, it is sufficient to switch in the algorithm the role played by the respective sets J_A and J_B, letting J_B be the pioneer set instead of J_A.

We denote by I the set of the current available positions in the optimal sequence and by I_A the initial index set in correspondence to the pioneer set J_A. Moreover, to take into account the role played by the parity of n_A and n_B, we introduce the binary switch variable s having the following meaning:

$$\begin{cases} s = 0 & \text{if both } n_A \text{ and } n_B \text{ are odd numbers,} \\ s = 1 & \text{otherwise.} \end{cases}$$

Then, given the job sets

$$J_A = \{a_1, \ldots, a_{n_A}\} \text{ and } J_B = \{b_1, \ldots, b_{n_B}\},$$

the algorithm can be stated as follows.

Algorithm 1.

Step 0 (Inizialization) If both n_A and n_B are odd numbers, set $s := 0$, otherwise set $s := 1$. Set $\bar{k} := \lfloor n/2 \rfloor - 1 + s$, $K := \{1, \ldots, \bar{k}\}$, $I := \{1, \ldots, n\}$ and $I_A := \{1, \ldots, n_A\}$. If $s = 0$ go to Step 1, otherwise go to Step 2.

Step 1 (Initial single assignment) Select an arbitrary index $i \in I_A$ and assign the job $a_i \in J_A$ to the position $n/2$. Set $I_A := I_A \setminus \{i\}$, $I := I \setminus \{n/2\}$ and go to Step 3.

Step 2 (Couple assignment) Select an arbitrary couple of indexes $i, j \in I_A$, with $i \neq j$, and an arbitrary index $k \in K$. Assign the job $a_i \in J_A$ to the position k and the job $a_j \in J_A$ to the position $n + 1 - k$. Set $I_A := I_A \setminus \{i, j\}$, $K := K \setminus \{k\}$ and $I := I \setminus \{k, n + 1 - k\}$.

Step 3 (Stopping criterion and assignment of J_B) If $I_A = \emptyset$, assign arbitrarily all the jobs of J_B to the remaining positions available in I and STOP; otherwise go to Step 2.

Some observations are in order. First of all, termination of the algorithm is trivially guaranteed due to the finiteness of the job sets J_A and J_B. Moreover we observe that, because of the assumption that J_A is the pioneer set, the algorithm switches between the following two cases: i) both n_A and n_B are odd numbers ($s = 0$); ii) n_A is even ($s = 1$). These two cases differ uniquely for the passage to Step 1, which accommodates a single job of J_A into the position $n/2$ and which is performed just once, only in the case i), where both n_A and n_B are odd and, consequently, n is even making this step well posed. Based on these considerations, Step 1 can be interpreted as the step aimed at making the cardinality of I_A even: in fact, after it is performed, the case i) reduces to the case ii).

Furthermore it is worth noting that the optimal solution provided by the algorithm is not uniquely determined, since it depends on some arbitrary choices at Step 2. In particular, apart from the selection of the couple of jobs in J_A to be accommodated into the optimal sequence at the current iteration, Step 2 is characterized also by the choice of the index k, which is performed $\lfloor \frac{n_A}{2} \rfloor$ times among $\lfloor \frac{n}{2} \rfloor + s - 1$ numbers without repetitions: consequently the number of obtainable different optimal solutions, in terms of positions assigned to J_A and J_B, is equal to the binomial coefficient

$$\binom{\lfloor \frac{n}{2} \rfloor + s - 1}{\lfloor \frac{n_A}{2} \rfloor}.$$

We will show the optimality of Algorithm 1 in the next section. To better clarify its behavior, in the following we report two examples considering the cases $s = 0$ and $s = 1$, respectively. We indicate by $\bar{C}_A$ and $\bar{C}_B$ the average completion times of the jobs in J_A and J_B, respectively.

Example 1. Letting $J_A = \{a_1, a_2, a_3\}$ and $J_B = \{b_1, b_2, b_3, b_4, b_5\}$, we have $n_A = 3$, $n_B = 5$, and, consequently, $n = 8$. The steps are the following.

Step 0: Since both n_A and n_B are odd, we set: $s := 0$. As a consequence $\bar{k} := 3$ and $K := \{1, 2, 3\}$. Moreover, $I := \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $I_A := \{1, 2, 3\}$. Since $s = 0$, we go to Step 1.

Step 1: Selecting $i = 1$, we have $[a_1] = 4$. Then $I_A := \{2, 3\}$, $K := \{1, 2, 3\}$, $I := \{1, 2, 3, 5, 6, 7, 8\}$ and we go to Step 3.

Step 3: Since $I_A \neq \emptyset$, we go to Step 2.

Step 2: We select $i = 2$, $j = 3$ and $k = 1$. As a consequence, $[a_2] = 1$ and $[a_3] = 8$. Then $I_A := \emptyset$, $K := \{2, 3\}$ and $I := \{2, 3, 5, 6, 7\}$.

Step 3: Since $I_A = \emptyset$, we assign all the jobs of J_B to the available positions in the set I and we stop.

The optimal sequence is the following:

$$\pi^* = \{a_2, b_1, b_2, a_1, b_3, b_4, b_5, a_3\}.$$

Then we have

$$\bar{C}_A = \frac{p + 4p + 8p}{3} = \frac{13}{3}p$$

and

$$\bar{C}_B = \frac{2p + 3p + 5p + 6p + 7p}{5} = \frac{23}{5}p.$$

Consequently $C^* = |\bar{C}_A - \bar{C}_B| = \frac{4}{15}p$.

Example 2. Letting $J_A = \{a_1, a_2, a_3, a_4\}$ and $J_B = \{b_1, b_2, b_3\}$, we have $n_A = 4$, $n_B = 3$, and, consequently, $n = 7$. Algorithm 1 works in the following way.

Step 0: Since n_A is even and n_B is odd, we set: $s := 1$. As a consequence $k := 3$ and $K := \{1, 2, 3\}$. Moreover, $I := \{1, 2, 3, 4, 5, 6, 7\}$ and $I_A := \{1, 2, 3, 4\}$. Since $s = 1$, we go to Step 2.
Step 2: We select $i = 1$, $j = 2$ and $k = 1$. As a consequence, $[a_1] = 1$ and $[a_2] = 7$. Then $I_A := \{3, 4\}$, $K := \{2, 3\}$ and $I := \{2, 3, 4, 5, 6\}$.
Step 3: Since $I_A \neq \emptyset$, we return to Step 2.
Step 2: We select $i = 3$, $j = 4$ and $k = 2$. As a consequence, $[a_3] = 2$ and $[a_4] = 6$. Then $I_A := \emptyset$, $K := \{3\}$ and $I := \{3, 4, 5\}$.
Step 3: Since $I_A = \emptyset$, we assign all the jobs of J_B to the available positions in the set I and we stop.

The optimal sequence is the following:

$$\pi^* = \{a_1, a_3, b_1, b_2, b_3, a_4, a_2\}.$$

Then we have

$$\bar{C}_A = \frac{p + 2p + 6p + 7p}{4} = 4p$$

and

$$\bar{C}_B = \frac{3p + 4p + 5p}{3} = 4p.$$

Consequently $C^* = |\bar{C}_A - \bar{C}_B| = 0$.

We observe that, differently from the second example, in the first one the optimal solution does not provide a perfect balancing between the average completion times of the job sets J_A and J_B: we will show in fact (see next section, Lemma 2) that, whenever both n_A and n_B are odd, then $C^* > 0$.

Remark 1. Apart from the case $s = 0$ where initially a job of J_A is placed in position $n/2$, if, at each iteration, the index k is chosen as the smallest one in the set K, the jobs $a_i, a_j \in J_A$ are assigned to the extreme positions of the optimal sequence, one at the beginning and the other one at the end: consequently, at the last iteration, the jobs belonging to J_B are accommodated into the central positions. In other words, an optimal assignment of the pioneer set J_A is identified by the following set $\bar{I}_A$ of positions:

$$\bar{I}_A = \left\{1, 2, \ldots, \left\lfloor \frac{n_A}{2} \right\rfloor, \frac{n}{2}, n - \left\lfloor \frac{n_A}{2} \right\rfloor + 1, n - \left\lfloor \frac{n_A}{2} \right\rfloor + 2, \ldots, n\right\} \quad (4)$$

if $s = 0$, and

$$\bar{I}_A = \left\{1, 2, \ldots, \left\lfloor \frac{n_A}{2} \right\rfloor, n - \left\lfloor \frac{n_A}{2} \right\rfloor + 1, n - \left\lfloor \frac{n_A}{2} \right\rfloor + 2, \ldots, n\right\} \quad (5)$$

if $s = 1$.

Based on (4) and (5), problem (2) is solvable in constant time; but, on the other hand, we have also to take into account that in the practical applications a complete solution of the problem requires to assign explicitly each job to exactly one position, which is necessarily performed at least in time $O(n)$ (see next section, Theorem 1).

4. Optimality

In this section we prove that the solution provided by Algorithm 1 is an optimal solution to problem (2), assuming J_A as the pioneer set (without loss of generality, as explained in Section 3), and we give a characterization of the optimal objective function value C^*.

We show first the following two lemmas.

Lemma 1. *Problem (2) is equivalent to the following problem:*

$$z^* \triangleq \min_{\pi} \left| \sum_{i=1}^{n_A} [a_i] - \frac{n_A(n+1)}{2} \right|, \quad (6)$$

with

$$z^* = \frac{C^* n_A n_B}{np}. \quad (7)$$

Proof. Multiplying the objective function of problem (2) by $n_A n_B$, we obtain

$$\min_{\pi} \left| n_B \sum_{i=1}^{n_A} C_{a_i} - n_A \sum_{i=1}^{n_B} C_{b_i} \right|. \quad (8)$$

By (3) we have:

$$\sum_{i=1}^{n_A} C_{a_i} + \sum_{i=1}^{n_B} C_{b_i} = p \sum_{i=1}^{n} i = \frac{pn(n+1)}{2}$$

and consequently:

$$\sum_{i=1}^{n_B} C_{b_i} = \frac{pn(n+1)}{2} - \sum_{i=1}^{n_A} C_{a_i}. \quad (9)$$

Substituting (9) in (8), we obtain:

$$\min_{\pi} \left| n_B \sum_{i=1}^{n_A} C_{a_i} - \frac{n_A pn(n+1)}{2} + n_A \sum_{i=1}^{n_A} C_{a_i} \right|,$$

i.e.

$$\min_{\pi} \left| n \sum_{i=1}^{n_A} C_{a_i} - \frac{n_A pn(n+1)}{2} \right|,$$

which, dividing by n, becomes:

$$\min_{\pi} \left| \sum_{i=1}^{n_A} C_{a_i} - \frac{n_A p(n+1)}{2} \right|,$$

or, equivalently:

$$\min_{\pi} \left| p \sum_{i=1}^{n_A} [a_i] - \frac{n_A p(n+1)}{2} \right|.$$

Finally, dividing by p, we obtain:

$$\min_{\pi} \left| \sum_{i=1}^{n_A} [a_i] - \frac{n_A(n+1)}{2} \right|.$$

As a consequence, relationship (7) holds. □

Remark 2. Problem (6) can be interpreted as a simplified version of problem (2), since it is independent on p and involves only the jobs of the pioneer set J_A. This consideration reflects the scheme of Algorithm 1, which works by first accommodating into the optimal sequence all the jobs of J_A and, successively, by arbitrarily assigning the jobs of J_B into the remaining positions.

Remark 3. As observed about problem (2), also the equivalent problem (6) appears as a particular case of the subset-sum problem. In fact, given the set $I = \{1, \ldots, n\}$, problem (6) reduces to find a subset $\bar{I} \subset I$, of cardinality n_A, such that the function

$$\left| \sum_{i \in \bar{I}} i - \frac{n_A(n+1)}{2} \right|$$

is minimized. As a consequence, it can be interpreted as a subset-sum problem whose objective is to minimize the deviation of the sum of the elements in $\bar{I}$, with respect to the target $T \triangleq \frac{n_A(n+1)}{2}$.

Lemma 2. *If both n_A and n_B are odd, the objective function of problem (6) takes the form*

$$x + \frac{1}{2},$$

with x being a nonnegative integer number. As a consequence, $C^* > 0$.

Proof. The objective function of problem (6) is

$$\left| \sum_{i=1}^{n_A} [a_i] - \frac{n_A(n+1)}{2} \right|.$$

Then, if both n_A and n_B are odd, n is even and $n_A(n+1)$ is odd; consequently, since the term

$$\sum_{i=1}^{n_A} [a_i]$$

is integer, the proof follows. Finally, taking into account (7), it holds $C^* > 0$. □

Theorem 1. *Algorithm 1 provides an optimal solution to problem (2), assigning each job exactly to one position in time $O(n)$.*

Proof. The calculation of the computational complexity is trivial. In fact, apart from the initialization step (Step 0), the core of the algorithm resides in Step 2 and Step 3, since Step 1 is eventually performed only once in time $O(1)$. More specifically, in time $O(n_A)$ the algorithm cycles between Step 2 and Step 3 for $\lfloor n_A/2 \rfloor$ times, accommodating each time two jobs of J_A into the optimal sequence, and terminates by assigning in time $O(n_B)$ all the jobs of J_B.

In order to show the optimality, we treat separately the two cases $s = 0$ and $s = 1$, respectively.

Case 1) $s = 0$: both n_A and n_B are odd.

By Lemma 1, problem (2) is equivalent to problem (6). The quantity $\sum_{i=1}^{n_A} [a_i]$, which corresponds to the sum of the positions assigned to the jobs of J_A, is easily computable on the basis of the following considerations. At Step 1 a job in J_A is scheduled in the position $n/2$, while at Step 2 two jobs in J_A are scheduled, for $\frac{n_A-1}{2}$ times, in the positions k and $n+1-k$, respectively. As a consequence:

$$\begin{aligned} \textstyle\sum_{i=1}^{n_A} [a_i] &= \frac{n}{2} + \frac{n_A - 1}{2}(k + n + 1 - k) \\ &= \frac{n_A(n+1) - 1}{2} \end{aligned} \tag{10}$$

Substituting (10) in the objective function of problem (6), we obtain

$$\left| \frac{n_A(n+1) - 1}{2} - \frac{n_A(n+1)}{2} \right| = \frac{1}{2},$$

which, by Lemma 2, is the optimal function value of problem (6).

Case 2) $s = 1$: n_A is even.

Analogously to the previous case, since at Step 2 two jobs in J_A are scheduled for $\frac{n_A}{2}$ times in the positions k and $n+1-k$ respectively, we have

$$\sum_{i=1}^{n_A} [a_i] = \frac{n_A(n+1)}{2}$$

and, by Lemma 1, taking into account the objective function of problem (6), it follows $z^* = 0$. □

Remark 4. Given C^*, by using formula (7) to compute the objective function value of problem (6), in correspondence to the optimal solutions of Example 1 and Example 2, we obtain, as expected, $z^* = \frac{1}{2}$ and $z^* = 0$, respectively.

Finally, from the proof of Theorem 1 and taking into account relationship (7), the following result holds.

Corollary 1. *Given problem (2), if at least one between n_A and n_B is even then $C^* = 0$, otherwise*

$$C^* = \frac{np}{2n_A n_B}.$$

5. A variant of the problem

In this section we face a variant of problem (2), in order to take into account more explicitly the cardinalities of J_A and J_B in the objective function.

In particular we tackle the following problem:

$$\begin{cases} C_w^* \triangleq \min\limits_{\pi} & \left| \dfrac{n_A}{n} \displaystyle\sum_{i=1}^{n_A} C_{a_i}(\pi) - \dfrac{n_B}{n} \sum_{i=i}^{n_B} C_{b_i}(\pi) \right| \\ & \text{with:} \\ & p_{a_i} = p \quad i = 1, \ldots, n_A \\ & p_{b_i} = p \quad i = 1, \ldots, n_B, \end{cases} \tag{11}$$

aimed at balancing the total weighted completion times of the two job sets, whose weights are the relative cardinalities n_A/n and n_B/n of J_A and J_B, respectively. It is easy to see that, analogously to problem (2), also problem (11) can be interpreted as a special case of the subset-sum problem.

For solving this problem we propose a mathematical programming formulation by introducing the following binary decision variables:

$$\begin{cases} x_{ij} = 1 & \text{if the job } a_i \text{ is processed in position } j, \\ x_{ij} = 0 & \text{otherwise,} \end{cases}$$

with $i = 1, \ldots, n_A$ and $j = 1, \ldots, n$, and

$$\begin{cases} y_{ij} = 1 & \text{if the job } b_i \text{ is processed in position } j, \\ y_{ij} = 0 & \text{otherwise,} \end{cases}$$

with $i = 1, \ldots, n_B$ and $j = 1, \ldots, n$. Then, recalling that $C_{[j]} = jp$ (see (3)), an optimization model solving problem (11) is the following:

$$\begin{cases} C_w^* = \min\limits_{x,y} & \left| \dfrac{n_A}{n} \displaystyle\sum_{i=1}^{n_A} \sum_{j=1}^{n} jpx_{ij} - \dfrac{n_B}{n} \sum_{i=1}^{n_B} \sum_{j=1}^{n} jpy_{ij} \right| \\ & \displaystyle\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \ldots, n_A \\ & \displaystyle\sum_{j=1}^{n} y_{ij} = 1 \quad i = 1, \ldots, n_B \\ & \displaystyle\sum_{i=1}^{n_A} x_{ij} + \sum_{i=1}^{n_B} y_{ij} = 1 \quad j = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \quad i = 1 \ldots, n_A \quad j = 1 \ldots, n \\ & y_{ij} \in \{0, 1\} \quad i = 1 \ldots, n_B \quad j = 1 \ldots, n, \end{cases} \tag{12}$$

where the first two groups of constraints impose that each job, in the sets J_A and J_B respectively, must be assigned exactly to one position, while the third group imposes that each position must be assigned exactly to one job.

The objective function of problem (12) is non-linear and it is a convex non-differentiable piecewise affine function. It is well known that such a problem can be linearized by introducing an additional decision variable, say v, resulting in the following equivalent mixed-integer linear program:

$$\begin{cases} \min\limits_{x,y,v} & pv \\ & v \geq \dfrac{n_A}{n} \displaystyle\sum_{i=1}^{n_A} \sum_{j=1}^{n} jx_{ij} - \dfrac{n_B}{n} \sum_{i=1}^{n_B} \sum_{j=1}^{n} jy_{ij} \\ & v \geq \dfrac{n_B}{n} \displaystyle\sum_{i=1}^{n_B} \sum_{j=1}^{n} jy_{ij} - \dfrac{n_A}{n} \sum_{i=1}^{n_A} \sum_{j=1}^{n} jx_{ij} \\ & \displaystyle\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \ldots, n_A \\ & \displaystyle\sum_{j=1}^{n} y_{ij} = 1 \quad i = 1, \ldots, n_B \\ & \displaystyle\sum_{i=1}^{n_A} x_{ij} + \sum_{i=1}^{n_B} y_{ij} = 1 \quad j = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \quad i = 1 \ldots, n_A \quad j = 1 \ldots, n \\ & y_{ij} \in \{0, 1\} \quad i = 1 \ldots, n_B \quad j = 1 \ldots, n, \end{cases} \tag{13}$$

which is characterized by one real variable, n^2 binary variables and $2n + 2$ constraints.

We note that, although problem (13) is an assignment type problem, it cannot be solved as a linear program, since the total unimodularity of the constraints matrix is lost, due to the presence of the first two constraints introduced to linearize the objective function of problem (12). On the other hand we observe that, when n_A and n_B coincide, problem (11) reduces to problem (2) and, consequently, it can be solved in constant time by using formulas (4) and (5), or in linear time by means of Algorithm 1 which provides in addition the job-position assignments.

In the sequel we will show that, when the difference between the cardinalities n_A and n_B of the two job sets J_A and J_B, respectively, is sufficiently large, the optimal unique solution consists in simply accommodating at the end of the sequence all the jobs, in any order, belonging to the set with the smallest cardinality and, consequently, in assigning to the first positions all the jobs, in any order, of the set with the largest cardinality.

Without loss of generality, in the following we consider J_A, the pioneer set, as the job set with the largest cardinality and J_B as the job set with the smallest cardinality. We prove first the following lemma.

Lemma 3. *Let $\bar{\pi} = \{a_1, \ldots, a_{n_A}, b_1, \ldots, b_{n_B}\}$. Then*

$$\frac{n_A}{n} \sum_{i=1}^{n_A} C_{a_i}(\bar{\pi}) - \frac{n_B}{n} \sum_{i=i}^{n_B} C_{b_i}(\bar{\pi}) \geq 0 \tag{14}$$

if and only if $n_B \leq \lfloor \bar{n}_A \rfloor$, with $\bar{n}_A \triangleq \dfrac{\sqrt{5n_A^2 + 6n_A + 1} - n_A - 1}{2}$.

Proof. By (3) we have:

$$\sum_{i=1}^{n_A} C_{a_i}(\bar{\pi}) = \frac{pn_A(n_A + 1)}{2} \tag{15}$$

and

$$\sum_{i=1}^{n_B} C_{b_i}(\bar{\pi}) = \frac{pn(n+1) - pn_A(n_A+1)}{2}. \quad (16)$$

Substituting (15) and (16) in (14), we obtain:

$$\frac{pn_A^2(n_A+1) - pn_Bn(n+1) + pn_Bn_A(n_A+1)}{2n} \geq 0, \quad (17)$$

which, multiplying by the positive quantity $2n/p$, becomes:

$$n_A^2(n_A+1) - n_Bn(n+1) + n_Bn_A(n_A+1) \geq 0.$$

Recalling that $n = n_A + n_B$, the above inequality can be rewritten as follows:

$$n_A^3 + n_A^2 - n_B^3 - 2n_B^2n_A - n_B^2 \geq 0,$$

i.e.

$$(n_A+n_B)(n_A^2 - n_An_B + n_A - n_B^2 - n_B) \geq 0.$$

Dividing by $n = n_A + n_B > 0$, we obtain:

$$n_A^2 - n_An_B + n_A - n_B^2 - n_B \geq 0, \quad (18)$$

whose solution, with respect to n_B, is:

$$\frac{-\sqrt{5n_A^2+6n_A+1} - n_A - 1}{2} \leq n_B \leq \frac{\sqrt{5n_A^2+6n_A+1} - n_A - 1}{2}.$$

The thesis follows recalling that n_B is a positive integer number. □

Theorem 2. *If $n_B \leq \lfloor \bar{n}_A \rfloor$, with*

$$\bar{n}_A \triangleq \frac{\sqrt{5n_A^2+6n_A+1} - n_A - 1}{2},$$

then the sequences characterized by all the jobs of J_A in the first positions (in any order) and all the jobs of J_B in the last positions (in any order) are the unique optimal solutions to problem (11), with

$$C_w^* = p\frac{n_A^3 - n_B^3 + n_A^2 - n_B^2 - 2n_B^2n_A}{2n}.$$

Proof. Let $\bar{\pi} = \{a_1, \ldots, a_{n_A}, b_1, \ldots, b_{n_B}\}$ be a sequence such that $n_B \leq \lfloor \bar{n}_A \rfloor$. First of all, we observe that all the sequences obtained from $\bar{\pi}$ by switching the positions between two jobs of J_A or between two jobs of J_B are characterized by the same objective function value provided by $\bar{\pi}$.

Then, denoting by $f(\pi)$ the objective function of problem (11), we show the optimality and the uniqueness by proving that, in correspondence to any other sequence $\hat{\pi} \neq \bar{\pi}$, obtained from $\bar{\pi}$ by switching the positions between a job of J_A and a job of J_B, it holds $f(\hat{\pi}) > f(\bar{\pi})$. In fact, letting

$$g(\pi) \triangleq \frac{n_A}{n}\sum_{i=1}^{n_A} C_{a_i}(\pi),$$

and

$$h(\pi) \triangleq \frac{n_B}{n}\sum_{i=i}^{n_B} C_{b_i}(\pi),$$

we have

$$f(\pi) = |g(\pi) - h(\pi)|.$$

Since $p > 0$, it holds:

$$g(\hat{\pi}) > g(\bar{\pi}) \quad (19)$$

and

$$h(\hat{\pi}) < h(\bar{\pi}). \quad (20)$$

By Lemma 3, we have:

$$f(\bar{\pi}) = g(\bar{\pi}) - h(\bar{\pi}). \quad (21)$$

From (21), taking into account inequalities (19) and (20), we obtain:

$$f(\hat{\pi}) = g(\hat{\pi}) - h(\hat{\pi}) > f(\bar{\pi}).$$

Consequently, $\bar{\pi}$ is optimal and

$$C_w^* = f(\bar{\pi}) = \frac{n_A}{n}\sum_{i=1}^{n_A} C_{a_i}(\bar{\pi}) - \frac{n_B}{n}\sum_{i=i}^{n_B} C_{b_i}(\bar{\pi}). \quad (22)$$

Substituting (15) and (16) in (22), and taking into account that $n = n_A + n_B$, we obtain:

$$C_w^* = p\frac{n_A^3 - n_B^3 + n_A^2 - n_B^2 - 2n_B^2n_A}{2n}. \quad \square$$

Remark 5. If $n_A \geq 2$, the value $n_B = \lceil \frac{n_A}{2} \rceil$ satisfies the inequality $n_B \leq \lfloor \bar{n}_A \rfloor$.

Remark 6. From Theorem 2 and taking into account (22), we have that, if $n_B \leq \lfloor \bar{n}_A \rfloor$, the function value in correspondence to any optimal solution π^* is:

$$C_w^* = \frac{n_A}{n}\sum_{i=1}^{n_A} C_{a_i}(\pi^*) - \frac{n_B}{n}\sum_{i=i}^{n_B} C_{b_i}(\pi^*).$$

Then, in such a case, the objective function of the mathematical programming model (12) can be assumed to be linear and equal to

$$\frac{n_A}{n}\sum_{i=1}^{n_A}\sum_{j=1}^{n} jpx_{ij} - \frac{n_B}{n}\sum_{i=1}^{n_B}\sum_{j=1}^{n} jpy_{ij},$$

making problem (12) an integer linear program with only assignment type constraints. As a consequence, by the total unimodularity of the constraints matrix, the continuous relaxation of problem (12), or equivalently of problem (13), provides an optimal solution to problem (11).

Declaration of competing interest

The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Acknowledgements

We would like to thank Manlio Gaudioso for the fruitful discussions and for his valuable suggestions.

References

[1] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems, fifth edition, Springer International Publishing, 2016.

[2] V. T'Kindt, J.-C. Billaut, Multicriteria Scheduling. Theory, Models and Algorithms, Springer, 2006.

[3] A. Agnetis, P.B. Mirchandani, D. Pacciarelli, A. Pacifici, Scheduling problems with two competing agents, Oper. Res. 52 (2) (2004) 229–242.

[4] T. Cheng, C.-Y. Liu, W.-C. Lee, M. Ji, Two-agent single-machine scheduling to minimize the weighted sum of the agents' objective functions, Comput. Ind. Eng. 78 (2014) 66–73.

[5] H. Li, Y. Gajpal, C. Bector, Single machine scheduling with two-agent for total weighted completion time objectives, Appl. Soft Comput. 70 (2018) 147–156.

[6] Q. Nong, T. Cheng, C. Ng, Two-agent scheduling to minimize the total cost, Eur. J. Oper. Res. 215 (1) (2011) 39–44.

[7] C.-C. Wu, W.-H. Wu, J.-C. Chen, Y. Yin, W.-H. Wu, A study of the single-machine two-agent scheduling problem with release times, Appl. Soft Comput. 13 (2) (2013) 998–1006.

[8] N. Huynh Tuong, A. Soukhal, J.-C. Billaut, Single-machine multi-agent scheduling problems with a global objective function, J. Sched. 15 (3) (2012) 311–321.

[9] J.Y. Leung, M. Pinedo, G. Wan, Competitive two-agent scheduling and its applications, Oper. Res. 58 (2) (2010) 458–469.

[10] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, Inc., New York, NY, USA, 1990.

EURO Journal on Computational Optimization 10 (2022) 100032

Contents lists available at ScienceDirect

EURO Journal on Computational Optimization

www.elsevier.com/locate/ejco

A Lagrangian heuristics for balancing the average weighted completion times of two classes of jobs in a single-machine scheduling problem

Matteo Avolio, Antonio Fuduli *

Department of Mathematics and Computer Science, University of Calabria, Rende, Italy

ARTICLE INFO

Keywords:
Scheduling
Single-machine
Weighted completion time
Lagrangian relaxation

ABSTRACT

We tackle a new single-machine scheduling problem, whose objective is to balance the average weighted completion times of two classes of jobs. Because both the job sets contribute to the same objective function, this problem can be interpreted as a cooperative two-agent scheduling problem, in contraposition to the standard multiagent problems, which are of the competitive type since each class of job is involved only in optimizing its agent's criterion. Balancing the completion times of different sets of tasks finds application in many fields, such as in logistics for balancing the delivery times, in manufacturing for balancing the assembly lines and in services for balancing the waiting times of groups of people.

To solve the problem, for which we show the NP-hardness, a Lagrangian heuristic algorithm is proposed. In particular, starting from a nonsmooth variant of the quadratic assignment problem, our approach is based on the Lagrangian relaxation of a linearized model and reduces to solve a finite sequence of successive linear assignment problems.

* Corresponding author.
E-mail addresses: matteo.avolio@unical.it (M. Avolio), antonio.fuduli@unical.it (A. Fuduli).

https://doi.org/10.1016/j.ejco.2022.100032

Numerical results are presented on a set of randomly generated test problems, showing the efficiency of the proposed technique.

1. Introduction

Processing two or more classes of jobs is the main characterization of the so-called multiagent scheduling problems [2], where various agents, sharing the same machines, process their own jobs, each one using different objective functions. The principal difference between multiobjective and multiagent scheduling problems resides in the fact that, while in the multiobjective case all the jobs contribute to every objective function, in the multiagent problems each class of jobs is involved in a single objective function, that one relative to the corresponding agent.

Two seminal papers on multiagent single-machine scheduling problems are [9] and [5]. The problems tackled in [9] are characterized by two or three agents and different combinations of following three objectives are considered: the minimization of maximum completion time, the minimization of the maximum lateness and the minimization of the weighted completion time. In [5] some two-agent scenarios are faced, choosing as objective functions the maximum of regular functions (associated with each job), the number of late jobs and the total weighted completion times. An extension of the latter problems to the case with more than two agents is reported in [6].

In the last years various works have focused on two-agent single-machine scheduling problems. In [4] a branch & bound algorithm has been designed, based on the computation of the bounds by means a Lagrangian relaxation technique, with the aim to minimize the total weighted completion time of the jobs of first agent, subject to a bound on the objective function of the second agent, consisting in the alternative three cases: the minimization of the total weighted completion time, the minimization of the maximum lateness and the minimization of the maximum completion time. In [25] a 2-approximation algorithm is presented to minimize the sum of two objectives, the maximum weighted completion time of the jobs of the first agent and the total weighted completion time of the jobs of the second agent. In [28] and [23] possible release times of the jobs are taken into account. In particular, in [28] the objective is to minimize the tardiness of the first agent, keeping the lateness of the second one below a certain level, while in [23] the linear combination of the makespans of both the agents is minimized.

Finally some recent works have been devoted also to the case of two-agent problems characterized by parallel machines, such as [13,22,29].

It is worth noting that all the above cited scheduling problems are characterized by the fact that the agents are in competition with each other, in the sense that every class

of jobs contributes only to the objective function of the corresponding agent. On the contrary, in this work we tackle a new scheduling problem, which could be interpreted as a cooperative type two-agent problem, since all the jobs are involved in the minimization of a common objective function, aimed at balancing the average weighted completion times of two classes of jobs, say J_A and J_B. Obtaining balanced solutions in a multiagent scheduling problem reminds the concept of *fairness* [11], which plays an important role in case an optimal solution, maximizing the system utility, is unacceptable by the worse-off agent. Recent works in this field are [3,8,24,30]: some of them concern the so-called *price of fairness* [10], which is a measure of how much the system utility has to be sacrificed in order to have a fair (balanced) solution.

The paper is organized as follows. In the next section we state formally the problem, showing its NP-hardness. In Section 3 we provide a formulation of the problem as a nonsmooth variant of the quadratic assignment problem and we focus on the Glover linearization of the proposed model, for which in Section 4 we propose a heuristic algorithm based on the Lagrangian relaxation. In Section 5 some numerical experiments are presented and some conclusions are drawn in Section 6.

Throughout the paper we use the following notation. We indicate by n_A and n_B the cardinalities of the job sets J_A and J_B, respectively. Letting $J \triangleq J_A \cup J_B$ and $n \triangleq n_A + n_B$, we denote, for each job $j \in J$, the processing time by p_j and the corresponding weight by w_j. Finally we indicate by π any job sequence and by $[j]$ the job processed in position j.

2. Problem definition and computational complexity

Given two job sets J_A and J_B with cardinality n_A and n_B, respectively, and using the Graham notation (see [19]), we tackle the following single-machine scheduling problem:

$$1 \mid\mid |\bar{C}_A - \bar{C}_B| \tag{1}$$

where

$$\bar{C}_A \triangleq \frac{\sum_{j \in J_A} w_j C_j}{n_A}$$

and

$$\bar{C}_B \triangleq \frac{\sum_{j \in J_B} w_j C_j}{n_B}$$

are the average weighted completion times of the jobs in J_A and J_B, respectively, with C_j being the completion time of job j.

Facing problem (1) has taken inspiration by the following practical problem, arisen in the academic context. In a recovery exam session, a professor (the machine) has to schedule the oral examinations of two groups of students (the jobs) belonging to two

different classes. Each student should be examined only on the parts of the syllabus where he/she failed during the regular exam session and then it is assumed that an estimate of the duration of the exam is known (and it is different) for everyone. Since the exam session is very short and, at the same time, there are a few available classrooms because of the lectures ongoing in the same period, he convokes all of them at the same time in the same classroom. If he would like to be impartial as much as possible toward the two classes, what is the best way to schedule the students with the aim of balancing the average waiting times of the two groups? We have modelled this problem as problem (1), introducing in addition the possibility to have a weight in correspondence to each job.

The necessity to balance the average completion times of two or more classes of jobs could find application in additional contexts, such as in logistics or manufacturing. Consider for example a freight transportation company, performing deliveries by a single drone. The shipping times depend on the depot-customer distance and mainly on the load of the corresponding cargo (which can be very different among the various shipments), the latter acting on the speed of the drone. Imagine that, in a given time period, the company has to serve two customers, each of them requiring various freights of different types and loads. Assume moreover that the drone can perform only one delivery at a time and the company wants to balance the average waiting times of the customers. This problem could be modelled as problem (1), with the drone being the machine and the deliveries to the two customers being J_A and J_B: for any job j the weight w_j is set equal to 1 and the processing time p_j is computed taking into account both the depot-customer distance and the load of the cargo.

Another example could be to balance the production of the semi-finished products, needed to assemble a finished product and characterized by high storage costs. Imagine the case where all the semi-finished products are manufactured by the same company owing a unique plant, which is able to perform in sequence different operations. Then all the semi-finished products are obtained by the same plant (single machine) performing in sequence, without precedence constraints, a finite number of independent different operations (jobs). Since the final product requires to assemble all the semi-products, in case the storage of the semi-products is expensive, a natural criterion to optimize the production could be to balance as much as possible the manufacture of the semi-products, by balancing the average completion times of the corresponding operations (more classes of jobs, one per semi-product), each of them characterized by a specific processing time.

Using the polynomial reduction from the well-known NP-complete partition problem [15], we show in the sequel that problem (1) is NP-hard in the weak sense. In [7] it has been proved that, whenever $w_j = 1$ and $p_j = p_{const}$ for any $j \in J$, the problem can be solved in linear time, or even in constant time if the job-position assignment procedure is not explicitly considered. It remains an open problem to show the eventual strong NP-hardness.

Throughout the paper, we indicate by C^* the optimal objective function value of problem (1) and by $C(\pi)$ the objective function value of problem (1) in correspondence to the job sequence π.

Partition problem. Given a set of positive integers $A \stackrel{\triangle}{=} \{a_1, \dots a_k\}$, let $K \stackrel{\triangle}{=} \{1, 2, \dots, k\}$ be the corresponding index set. Is there a partition of K into two subsets K_1 and K_2 such that $\sum_{j \in K_1} a_j = \sum_{j \in K_2} a_j$?

Theorem 1. *Problem (1) is NP-hard.*

Proof. Given an instance $A = \{a_1, \dots, a_k\}$ of the partition problem, we construct an instance of problem (1), by setting:

$$\begin{cases} J_A = \{1, \dots, k\}; \quad J_B = \{k+1\}; \\ p_j = a_j, \ j = 1, \dots, k; \quad p_{k+1} = 1; \\ w_1 = 1; \quad w_j = 0, \ j = 2, \dots, k; \\ w_{k+1} = \frac{\sum_{j=1}^{k} a_j}{2k(\sum_{j=1}^{k} a_j + 1)}. \end{cases} \tag{2}$$

We show that the partition problem has a solution if and only if, in correspondence to the instance (2), problem (1) admits an optimal sequence π^* with $C^* = 0$.

($\Rightarrow$) Suppose that the partition problem admits a solution, i.e. there exists a partition of K into two subsets K_1 and K_2 such that $\sum_{j \in K_1} a_j = \sum_{j \in K_2} a_j$. Then

$$\sum_{j \in K_1} a_j = \sum_{j \in K_2} a_j = \frac{\sum_{j=1}^{k} a_j}{2}. \tag{3}$$

Without loss of generality, assume that $1 \in K_1$ and let $\bar{K}_1 \stackrel{\triangle}{=} K_1 \setminus \{1\}$. In case $1 \in K_2$, it is sufficient to switch the roles played by K_1 and K_2. Then the sequence $\pi^* = (\bar{K}_1, 1, K_2, k+1)$, where the jobs in $\bar{K}_1$ and K_2 are scheduled in any order, is an optimal solution to problem (1) with $C^* = 0$. In fact, taking into account (3) and recalling that $w_1 = 1$ and $w_j = 0$ for $j = 2, \dots, k$, we have

$$\bar{C}_A = \frac{\sum_{j \in J_A} w_j C_j}{n_A} = \frac{C_1}{k} = \frac{\sum_{j=1}^{k} a_j}{2k}$$

and

$$\bar{C}_B = \frac{\sum_{j \in J_B} w_j C_j}{n_B} = w_{k+1} C_{k+1} = \frac{\sum_{j=1}^{k} a_j}{2k(\sum_{j=1}^{k} a_j + 1)} (\sum_{j=1}^{k} a_j + 1) = \frac{\sum_{j=1}^{k} a_j}{2k}.$$

Then $|\bar{C}_A - \bar{C}_B| = 0$ and, consequently, π^* is an optimal solution to problem (1).

($\Leftarrow$) Suppose that, in correspondence to the instance (2), problem (1) admits an optimal sequence π^* such that $C^* = 0$. Then

$$\bar{C}_A = \bar{C}_B,$$

i.e.

$$\frac{C_1}{k} = w_{k+1} C_{k+1}.$$

Substituting the value of w_{k+1} in the above equality, we obtain:

$$C_1 = \frac{C_{k+1} \sum_{j=1}^{k} a_j}{2(\sum_{j=1}^{k} a_j + 1)},$$

i.e.

$$C_{k+1} = 2C_1 + \frac{2C_1}{\sum_{j=1}^{k} a_j}. \tag{4}$$

Since the processing times in the instance (2) are integer, then all the completion times are integer. As a consequence also the second term of the right hand side of (4) is integer, i.e. there exists an integer $t \geq 0$ such that

$$2C_1 = t \sum_{j=1}^{k} a_j. \tag{5}$$

The case $t = 0$ cannot occur because $C_1 > 0$. Moreover, in case $t \geq 2$, from (5) we would have

$$C_1 \geq \sum_{j=1}^{k} a_j$$

and, from (4), it would follow:

$$C_{k+1} \geq 2 \sum_{j=1}^{k} a_j + 2,$$

which is impossible since each completion time cannot exceed the sum of all the processing times, which is equal to $\sum_{j=1}^{k} a_j + 1$. Then in (5) we have necessarily $t = 1$, i.e.

$$C_1 = \frac{\sum_{j=1}^{k} a_j}{2}. \tag{6}$$

In summary, we have shown that, whenever $C^* = 0$ in correspondence to the instance (2), in any optimal sequence the completion time of job 1 is given by formula (6). As a consequence, comparing (3) and (6), a solution to the partition problem is obtained by setting K_1 (or indifferently K_2) equal to $\{1\} \cup \bar{K}_1$, where $\bar{K}_1$ contains all and only the indexes corresponding to the jobs which, in any optimal sequence, precede job 1. □

3. Problem formulation and Glover linearization

For solving problem (1) we propose a heuristic Lagrangian relaxation based approach, stemming from the mathematical formulation of the problem as a nonsmooth variant of the well known quadratic assignment problem (see [12] for an extensive review).

We define, for $j \in J$ and $t = 1, \ldots, n$, the following decision variables:

$$x_{jt} \triangleq \begin{cases} 1 & \text{if job } j \text{ is assigned to position } t \\ 0 & \text{otherwise,} \end{cases}$$

grouped, for the sake of simplicity, into the vector x of dimension n^2.

Taking into account that the completion time of a job j scheduled in the position t is

$$p_j + \sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li},$$

problem (1) can be formulated as follows:

$$\begin{cases} C^* = \min\limits_{x} \left| \dfrac{1}{n_A} \left[\displaystyle\sum_{j \in J_A} w_j \left(p_j + \sum_{t=1}^{n} \left(\sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \right) x_{jt} \right) \right] \right. \\ \qquad \left. - \dfrac{1}{n_B} \left[\displaystyle\sum_{j \in J_B} w_j \left(p_j + \sum_{t=1}^{n} \left(\sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \right) x_{jt} \right) \right] \right| & \text{(a)} \\ \displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J & \text{(b)} \\ \displaystyle\sum_{j \in J} x_{jt} = 1 \quad t = 1, \ldots, n & \text{(c)} \\ x_{jt} \in \{0, 1\} \quad j \in J, \quad t = 1, \ldots, n, & \text{(d)} \end{cases} \tag{7}$$

where the constraints (7b)-(7d) are the classical assignment constraints, which impose that each job must be assigned to exactly one position and each position to exactly one job.

Letting $p_A \triangleq \sum_{j\in J_A} w_j p_j$ and $p_B \triangleq \sum_{j\in J_B} w_j p_j$, problem (7) is easily rewritable as follows:

$$\begin{cases} C^* = \min\limits_{x} \left| \dfrac{1}{n_A}\left(p_A + \sum\limits_{j\in J_A}\sum\limits_{t=1}^{n} w_j x_{jt} \sum\limits_{l\in J}\sum\limits_{i=1}^{t-1} p_l x_{li}\right) \right. \\ \qquad\qquad \left. - \dfrac{1}{n_B}\left(p_B + \sum\limits_{j\in J_B}\sum\limits_{t=1}^{n} w_j x_{jt} \sum\limits_{l\in J}\sum\limits_{i=1}^{t-1} p_l x_{li}\right)\right| \\ \qquad\qquad \text{assignment constraints (7b)-(7d),} \end{cases}$$

which is a sort of quadratic assignment problem, characterized by a nonsmooth objective function. By introducing the auxiliary variable v, it can be rewritten in the following equivalent form, corresponding to a mixed integer quadratically constrained program (MIQCP):

$$\begin{cases} C^* = \min\limits_{x,v}\ v \\ \qquad v \geq \dfrac{1}{n_A}\left(p_A + \sum\limits_{j\in J_A}\sum\limits_{t=1}^{n} w_j x_{jt} \sum\limits_{l\in J}\sum\limits_{i=1}^{t-1} p_l x_{li}\right) \\ \qquad\quad - \dfrac{1}{n_B}\left(p_B + \sum\limits_{j\in J_B}\sum\limits_{t=1}^{n} w_j x_{jt} \sum\limits_{l\in J}\sum\limits_{i=1}^{t-1} p_l x_{li}\right) \\ \qquad v \geq \dfrac{1}{n_B}\left(p_B + \sum\limits_{j\in J_B}\sum\limits_{t=1}^{n} w_j x_{jt} \sum\limits_{l\in J}\sum\limits_{i=1}^{t-1} p_l x_{li}\right) \\ \qquad\quad - \dfrac{1}{n_A}\left(p_A + \sum\limits_{j\in J_A}\sum\limits_{t=1}^{n} w_j x_{jt} \sum\limits_{l\in J}\sum\limits_{i=1}^{t-1} p_l x_{li}\right) \\ \qquad \text{assignment constraints (7b)-(7d).} \end{cases} \tag{8}$$

A possible way to treat the nonlinearities of problem (8) is to use the Glover linearization [18], as shown in the following theorem.

Theorem 2. *Letting*

$$d_t \stackrel{\triangle}{=} (t-1)P, \quad t = 1, \ldots, n,$$

with

$$P \stackrel{\triangle}{=} \sum_{l \in J} p_l,$$

problem (8) is equivalent to the following mixed integer linear program (MILP):

$$\begin{cases} C^* = \min\limits_{x,v,z} v & \text{(a)} \\ v \geq \dfrac{1}{n_A}\left(p_A + \sum\limits_{j \in J_A}\sum\limits_{t=1}^{n} w_j z_{jt}\right) - \dfrac{1}{n_B}\left(p_B + \sum\limits_{j \in J_B}\sum\limits_{t=1}^{n} w_j z_{jt}\right) & \text{(b)} \\ v \geq \dfrac{1}{n_B}\left(p_B + \sum\limits_{j \in J_B}\sum\limits_{t=1}^{n} w_j z_{jt}\right) - \dfrac{1}{n_A}\left(p_A + \sum\limits_{j \in J_A}\sum\limits_{t=1}^{n} w_j z_{jt}\right) & \text{(c)} \\ d_t x_{jt} + \sum\limits_{l \in J}\sum\limits_{i=1}^{t-1} p_l x_{li} - z_{jt} \leq d_t \quad j \in J, \quad t = 1 \ldots, n & \text{(d)} \\ z_{jt} \leq \sum\limits_{l \in J}\sum\limits_{i=1}^{t-1} p_l x_{li} \quad j \in J, \quad t = 1 \ldots, n & \text{(e)} \\ z_{jt} \leq d_t x_{jt} \quad j \in J, \quad t = 1 \ldots, n & \text{(f)} \\ z_{jt} \geq 0 \quad j \in J, \quad t = 1 \ldots n & \text{(g)} \\ \text{assignment constraints (7b)-(7d).} & \text{(h)} \end{cases} \tag{9}$$

Proof. Problem (8) can be easily rewritten as follows:

$$
\begin{cases}
C^* = \min\limits_{x,v,z} v \\
\quad v \geq \dfrac{1}{n_A}\left(p_A + \displaystyle\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}\right) - \dfrac{1}{n_B}\left(p_B + \displaystyle\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}\right) \\
\quad v \geq \dfrac{1}{n_B}\left(p_B + \displaystyle\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}\right) - \dfrac{1}{n_A}\left(p_A + \displaystyle\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}\right) \\
\quad z_{jt} = x_{jt}\displaystyle\sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} \quad j \in J, \quad t = 1\dots n \\
\quad \text{assignment constraints (7b)-(7d)},
\end{cases}
$$

where, for $j \in J$ and $t = 1 \dots n$, the nonlinear constraints

$$z_{jt} = x_{jt}\sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} \tag{10}$$

impose that

$$x_{jt} = 1 \quad \Rightarrow \quad z_{jt} = \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} \tag{11}$$

and

$$x_{jt} = 0 \quad \Rightarrow \quad z_{jt} = 0. \tag{12}$$

Since

$$d_t = (t-1)P = \sum_{l\in J}\sum_{i=1}^{t-1} p_l, \tag{13}$$

it is easy to show that, in problem (9) the linear constraints (9d)-(9g) guarantee the satisfaction of both the implications (11) and (12). □

Problem (9) constitutes the Glover linearization of problem (8). Such linearization is obtained by introducing n^2+1 continuous variables and $4n^2+2$ constraints, including the nonnegativity of z_{jt}, which expresses the time that job j starts to be processed, when it is assigned to the position t (see (10)). In passing, we recall that, in the case of a standard quadratic assignment problem characterized by positive cost coefficients in the objective function, the Glover linearization reduces to the Kaufmann and Broeckx linearization [21], which requires n^2 new continuous variables and $2n^2$ additional constraints.

We conclude this section by showing that, under mild assumptions, the optimal objective function value of the continuous relaxation of problem (9) provides a null lower bound.

Lemma 1. *Let*

$$\frac{p_B}{n_B} \geq \frac{p_A}{n_A}. \tag{14}$$

If

$$P \geq \frac{2(p_B n_A - p_A n_B)}{n_B(n-1)w_A}, \tag{15}$$

with $w_A \triangleq \sum_{j \in J_A} w_j$*, then the optimal objective function value of the continuous relaxation of problem (9) is equal to zero.*

Proof. We show that the solution

$$\begin{cases} x_{jt} = 1/n & j \in J, \quad t = 1, \ldots, n; \qquad \text{(a)} \\ z_{jt} = \dfrac{2(t-1)(p_B n_A - p_A n_B)}{n(n-1)n_B w_A} & j \in J_A, \quad t = 1, \ldots, n; \qquad \text{(b)} \\ z_{jt} = 0 & j \in J_B, \quad t = 1, \ldots, n \qquad \text{(c)} \end{cases} \tag{16}$$

is feasible for the continuous relaxation of problem (9) and provides a null optimal value of the objective function v.

First of all, we trivially observe that vector x, defined by (16a), satisfies the assignment constraints (7b)-(7c) and, in addition, $0 \leq x \leq e$, where e is the vector of ones. Moreover, by (14), we have $z_{jt} \geq 0$ for $j \in J$ and $t = 1, \ldots, n$.

Taking into account (13) and (16a) and since $n \geq 2$, constraints (9d) are satisfied by any nonnegative values of z_{jt}.

As for constraints (9e) and (9f), it is easy to observe that, by definition of d_t, they coincide in correspondence to the values $x_{jt} = 1/n$. Then, to show their satisfaction, it is sufficient to prove that

$$z_{jt} \leq d_t/n,$$

for $j \in J$ and $t = 1, \ldots, n$. For $j \in J_B$ and $t = 1, \ldots, n$, the above inequality is trivially satisfied because of the nonnegativity of d_t and n. As for the indexes $j \in J_A$ and $t = 1, \ldots, n$, combining (15) and (16b) and recalling the definition of d_t, we have:

$$z_{jt} = \frac{2(t-1)(p_B n_A - p_A n_B)}{n(n-1)n_B w_A} \leq \frac{(t-1)P}{n} = \frac{d_t}{n}.$$

About the optimality, constraints (9b) and (9c), together with the objective function (9a), guarantee that, in correspondence to any optimal solution, it holds:

$$v = \left| \frac{1}{n_A} \left(p_A + \sum_{j \in J_A} \sum_{t=1}^{n} w_j z_{jt} \right) - \frac{1}{n_B} \left(p_B + \sum_{j \in J_B} \sum_{t=1}^{n} w_j z_{jt} \right) \right|.$$

Substituting in the above equality the values of z_{jt} given by (16b) and (16c), we have:

$$\begin{aligned} v &= \left| \frac{1}{n_A} \sum_{j \in J_A} \sum_{t=1}^{n} w_j \frac{2(t-1)(p_B n_A - p_A n_B)}{n(n-1) n_B w_A} + \frac{p_A}{n_A} - \frac{p_B}{n_B} \right| \\ &= \left| \frac{w_A}{n_A} \frac{n(n-1)}{2} \frac{2(p_B n_A - p_A n_B)}{n(n-1) n_B w_A} + \frac{p_A}{n_A} - \frac{p_B}{n_B} \right| \\ &= \left| \frac{p_B n_A - p_A n_B}{n_A n_B} + \frac{p_A}{n_A} - \frac{p_B}{n_B} \right| \\ &= 0. \quad \square \end{aligned}$$

Remark 1. Lemma 1 holds also when condition (14) is not satisfied. In such a case, it is sufficient to swap the roles played by J_A and J_B and condition (15) becomes

$$P \geq \frac{2(p_A n_B - p_B n_A)}{n_A (n-1) w_B}, \tag{17}$$

where $w_B \stackrel{\triangle}{=} \sum_{j \in J_B} w_j$.

Remark 2. It is easy to verify that, in the unweighted case, i.e. when $w_j = 1$ for any $j \in J$, since $w_A = n_A$ and $P = p_A + p_B$, condition (15) is automatically satisfied, provided that $n \geq 3$. This condition is apparently a mild assumption even in the weighted case, as confirmed by its satisfaction in correspondence to the more than one thousand randomly generated test problems, used for our numerical experiments (see Section 5).

4. A Lagrangian relaxation based approach

Problem (9) is a mixed integer linear program characterized by n^2 integer variables, $n^2 + 1$ continuous variables and $4n^2 + 2n + 2$ constraints (including the nonnegativity of z_{jt}). As a consequence, solving it exactly requires a remarkable effort from the computational point of view, especially for large values of n.

Then, for solving problem (1) we propose a Lagrangian relaxation approach (see the surveys [14,16,20]), based on relaxing all the constraints of problem (9), except the assignment ones and the nonnegativity constraints on variables z_{jt}. The motivation is

to design a heuristic algorithm requiring to solve, at each iteration, a linear assignment problem.

4.1. The Lagrangian relaxation problem

Dualizing constraints (9b)-(9f) and grouping all the corresponding Lagrangian multipliers into the vector $\lambda \in \mathbb{R}^{3n^2+2}$, the Lagrangian relaxation problem of the mixed integer linear program (9) is the following:

$$LR(\lambda) \begin{cases} f_{LR}(\lambda) \stackrel{\triangle}{=} \min\limits_{x,v,z} f(x,v,z) \\ \\ z_{jt} \geq 0 \quad j \in J, \quad t = 1, \ldots, n \\ \\ \text{assignment constraints (7b)-(7d)}, \end{cases}$$

where $\lambda \geq 0$ and

$$\begin{aligned} f(x,v,z) \stackrel{\triangle}{=} v &+ \left[\frac{\alpha}{n_A}\left(\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt} + p_A\right)\right] - \left[\frac{\alpha}{n_B}\left(\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt} + p_B\right)\right] - \alpha v \\ &+ \left[\frac{\beta}{n_B}\left(\sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt} + p_B\right)\right] - \left[\frac{\beta}{n_A}\left(\sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt} + p_A\right)\right] - \beta v \\ &+ \sum_{j\in J}\sum_{t=1}^{n} \gamma_{jt}\left(d_t x_{jt} + \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li} - z_{jt} - d_t\right) \\ &+ \sum_{j\in J}\sum_{t=1}^{n} \delta_{jt}\left(z_{jt} - \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}\right) \\ &+ \sum_{j\in J}\sum_{t=1}^{n} \epsilon_{jt}\left(z_{jt} - d_t x_{jt}\right), \end{aligned}$$

with α, β, γ_{jt}, δ_{jt}, ϵ_{jt}, $j \in J$ and $t = 1, \ldots, n$, being the Lagrangian multipliers (grouped in vector λ) corresponding to the constraints (9b)-(9f), respectively.

The above function f can be arranged in the following form:

$$f(x,v,z) = f_x(x) + f_v(v) + f_z(z) + K,$$

where

$$f_x(x) = \sum_{j\in J}\sum_{t=1}^{n}\gamma_{jt}d_t x_{jt} + \sum_{j\in J}\sum_{t=1}^{n}\gamma_{jt}\sum_{l\in J}\sum_{i=1}^{t-1}p_l x_{li}$$

$$-\sum_{j\in J}\sum_{t=1}^{n}\delta_{jt}\sum_{l\in J}\sum_{i=1}^{t-1}p_l x_{li} - \sum_{j\in J}\sum_{t=1}^{n}\epsilon_{jt}d_t x_{jt},$$

$$f_v(v) = v(1-\alpha-\beta),$$

$$f_z(z) = \sum_{j\in J_A}\sum_{t=1}^{n} z_{jt}\left(\frac{w_j\alpha}{n_A} - \frac{w_j\beta}{n_A} - \gamma_{jt} + \delta_{jt} + \epsilon_{jt}\right)$$

$$+\sum_{j\in J_B}\sum_{t=1}^{n} z_{jt}\left(\frac{w_j\beta}{n_B} - \frac{w_j\alpha}{n_B} - \gamma_{jt} + \delta_{jt} + \epsilon_{jt}\right)$$

and

$$K = \frac{p_A\alpha}{n_A} - \frac{p_B\alpha}{n_B} + \frac{p_B\beta}{n_B} - \frac{p_A\beta}{n_A} - \sum_{j\in J}\sum_{t=1}^{n} d_t\gamma_{jt}. \tag{18}$$

As a consequence, the Lagrangian problem $LR(\lambda)$ is a separable program and, for any value of $\lambda \geq 0$, it can be solved by separately solving the following linear programs:

$$LR_x(\lambda)\begin{cases} f_{LR_x}(\lambda) \triangleq \min\limits_x f_x(x) \\ \text{assignment constraints (7b)-(7d)}, \end{cases}$$

$$LR_v(\lambda)\begin{cases} f_{LR_v}(\lambda) \triangleq \min\limits_v f_v(v) \\ 0 \leq v \leq \bar{v}, \end{cases}$$

and

$$LR_z(\lambda)\begin{cases} f_{LR_z}(\lambda) \triangleq \min\limits_z f_z(z) \\ 0 \leq z_{jt} \leq d_t, \quad j \in J, \quad t = 1,\ldots,n. \end{cases}$$

Some comments about the above problems are in order. Problem $LR_x(\lambda)$ is a linear assignment problem, for which it is well known that constraints $x_{jt} \in \{0,1\}$ can be substituted by constraints $x_{jt} \geq 0$.

In problem $LR_v(\lambda)$, since variable v represents the original objective function of problem (1), then such variable is nonnegative and bounded from above by any objective function value $\bar{v}$, computed in correspondence to any arbitrary job sequence in the set J.

In problem $LR_z(\lambda)$ any variable z_{jt}, $j \in J$ and $t = 1,\ldots,n$, is bounded from above by d_t, as a consequence of constraints (9f) and (7d).

It is worth noting that problems $LR_v(\lambda)$ and $LR_z(\lambda)$ are easily solvable by inspection. In fact, indicating by $v(\lambda)$ and $z_{jt}(\lambda)$, $j \in J$ and $t = 1, \ldots, n$, the respective optimal solutions, we have:

$$v(\lambda) = \begin{cases} 0 & \text{if } \alpha + \beta \leq 1 \\ \bar{v} & \text{otherwise,} \end{cases}$$

$$z_{jt}(\lambda) = \begin{cases} 0 & \text{if } \dfrac{w_j(\alpha - \beta)}{n_A} \geq \gamma_{jt} - \delta_{jt} - \epsilon_{jt} \\ d_t & \text{otherwise} \end{cases}$$

for $j \in J_A$ and $t = 1, \ldots, n$, and

$$z_{jt}(\lambda) = \begin{cases} 0 & \text{if } \dfrac{w_j(\beta - \alpha)}{n_B} \geq \gamma_{jt} - \delta_{jt} - \epsilon_{jt} \\ d_t & \text{otherwise} \end{cases}$$

for $j \in J_B$ and $t = 1, \ldots, n$.

Then, for any $\lambda \geq 0$, we come out with:

$$f_{LR}(\lambda) = f_{LR_x}(\lambda) + f_{LR_v}(\lambda) + f_{LR_z}(\lambda) + K, \tag{19}$$

which constitutes a lower bound on the optimal objective function value C^* of problem (9), i.e.

$$f_{LR}(\lambda) \leq C^*, \quad \text{for any } \lambda \geq 0.$$

We conclude the subsection by providing the following theorem, which gives a characterization of any lower bound $f_{LR}(\lambda)$ in case Lemma 1 holds.

Theorem 3. *If*

$$P \geq 2 \max \left\{ \frac{p_B n_A - p_A n_B}{n_B(n-1)w_A}, \frac{p_A n_B - p_B n_A}{n_A(n-1)w_B} \right\}, \tag{20}$$

then $f_{LR}(\lambda) \leq 0$ *for any* $\lambda \geq 0$.

Proof. Let

$$LD \left\{ f_{LR}^* \triangleq \max_{\lambda \geq 0} f_{LR}(\lambda) \right.$$

be the Lagrangian dual of problem (9). Then, for any $\lambda \geq 0$, it holds

$$f_{LR}(\lambda) \leq f_{LR}^*.$$

Because of the integrality property (see [14,16,20]), f^*_{LR} coincides with the objective function value of the continuous relaxation of problem (9). By (20), taking into account Lemma 1 and Remark 1, such value is equal to zero. □

4.2. The heuristic Lagrangian algorithm

We have all the ingredients to design a Lagrangian relaxation algorithm for solving problem (1).

The core of our approach is to generate, at each iteration for any fixed nonnegative value of λ, a trial solution $x(\lambda)$ to problem (7), or equivalently to problem (9), by solving the linear assignment problem $LR_x(\lambda)$.

Since a feasible solution of the scheduling problem trivially requires only the job-position assignments, we do not need, differently from a general Lagrangian relaxation framework, a feasibility recovering procedure to adjust the current solution in case the relaxed constraints (9b)-(9f), aimed at guarantee the optimality, are not satisfied. Instead, once a trial solution is generated, we try to improve it by performing a local search (see Subsection 4.3).

In Algorithm 1 we summarize our Lagrangian heuristic approach, where by L and U we denote, respectively, the lower bound and the upper bound (the incumbent), available at the current iteration on the optimal objective function value C^*. Moreover, for any fixed value of λ, using the notation introduced at the end of Section 1, we indicate by $\pi(\lambda)$ the sequence generated by $x(\lambda)$, such that $[t] = j$ in correspondence to $x_{jt}(\lambda) = 1$. We denote also by $\partial f_{LR}(\lambda)$ the subdifferential of function f_{LR} at λ (see for example the fresh survey [17] on nonsmooth optimization) and by $\|\cdot\|$ the Euclidean norm.

The steps of the algorithm reflect a standard Lagrangian relaxation scheme aimed at shrinking, at each iteration, the interval $[L, U]$ containing C^*. We stop the iterative procedure either after a prefixed time limit, or when a maximum number k_{max} of iterations is reached, or when the difference between U and L becomes less than or equal to a certain threshold.

At step 7, to possibly decrease the incumbent, we perform a local search (see for example [1]), which will be object of the next subsection. The best sequence π_λ provided by the local searches constitutes a heuristic solution to problem (1).

At steps 18-19, we update the vector λ of the multipliers by a standard subgradient method [27], aimed at solving the nonsmooth Lagrangian dual problem LD and where the projection on the nonnegative orthant guarantees the nonnegativity of λ. As concerns the computation of a subgradient g of f_{LR} at λ, we take

$$g = \begin{bmatrix} & g_\alpha & \\ & g_\beta & \\ g_{\gamma_{jt}} & j \in J, \quad t = 1, \dots, n \\ g_{\delta_{jt}} & j \in J, \quad t = 1, \dots, n \\ g_{\epsilon_{jt}} & j \in J, \quad t = 1, \dots, n \end{bmatrix},$$

Algorithm 1: Lagrangian heuristics.

Input: a sequence $\bar{\pi}_{init}$; $\lambda \geq 0$
Output: a sequence $\bar{\pi}$

▷Initialization
$\bar{\pi} \leftarrow \bar{\pi}_{init}$
$\bar{v} \leftarrow C(\bar{\pi}_{init})$
$L \leftarrow -\infty$
$U \leftarrow \bar{v}$
repeat
  ▷Computing a trial solution
  Solve problem $LR_x(\lambda)$ to compute $x(\lambda)$ and $f_{LR_x}(\lambda)$
  ▷Local search
  Starting from $\pi(\lambda)$, compute π_λ by a local search
  ▷Updating the incumbent
  Compute $C(\pi_\lambda)$
  if $C(\pi_\lambda) < U$ **then**
    $U \leftarrow C(\pi_\lambda)$
    $\bar{\pi} \leftarrow \pi_\lambda$
  ▷Updating the lower bound
  Solve problem $LR_v(\lambda)$ to compute $f_{LR_v}(\lambda)$
  Solve problem $LR_z(\lambda)$ to compute $f_{LR_z}(\lambda)$
  Compute K // see formula (18)
  $f_{LR}(\lambda) \leftarrow f_{LR_x}(\lambda) + f_{LR_v}(\lambda) + f_{LR_z}(\lambda) + K$
  if $f_{LR}(\lambda) > L$ **then**
    $L \leftarrow f_{LR}(\lambda)$
  ▷Updating the multipliers
  Compute $t > 0$ and $g \in \partial f_{LR}(\lambda)$
  $\lambda \leftarrow \max\{0, \lambda + t\frac{g}{\|g\|}\}$
until *a stopping criterion is satisfied*

where

$$g_\alpha = \frac{1}{n_A}\left(p_A + \sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}(\lambda)\right) - \frac{1}{n_B}\left(p_B + \sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}(\lambda)\right) - v(\lambda),$$

$$g_\beta = \frac{1}{n_B}\left(p_B + \sum_{j\in J_B}\sum_{t=1}^{n} w_j z_{jt}(\lambda)\right) - \frac{1}{n_A}\left(p_A + \sum_{j\in J_A}\sum_{t=1}^{n} w_j z_{jt}(\lambda)\right) - v(\lambda),$$

$$g_{\gamma_{jt}} = d_t x_{jt}(\lambda) + \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}(\lambda) - z_{jt}(\lambda) - d_t,$$

$$g_{\delta_{jt}} = z_{jt}(\lambda) - \sum_{l\in J}\sum_{i=1}^{t-1} p_l x_{li}(\lambda)$$

and

$$g_{\epsilon_{jt}} = z_{jt}(\lambda) - d_t x_{jt}(\lambda),$$

with $x_{jt}(\lambda)$, $j \in J$ and $t = 1, \ldots, n$, $v(\lambda)$ and $z_{jt}(\lambda)$, $j \in J$ and $t = 1, \ldots, n$, being the optimal solutions of the respective relaxed subproblems $LR_x(\lambda)$, $LR_v(\lambda)$ and $LR_z(\lambda)$.

Finally the stepsize t is the Polyak stepsize [26], given by:

$$t = \frac{f^*_{LR} - L}{\|g\|}. \tag{21}$$

Note that, in case Theorem 3 holds, f^*_{LR} is equal to zero, providing $t = -L/\|g\|$, with $L \leq 0$. Vice-versa, in case f^*_{LR} is unknown, its value in formula (21) can be substituted by the incumbent or, because of the integrality property, can be computed by initially solving the continuous relaxation of problem (9).

4.3. The local search

A relevant role in Algorithm 1 is played by step 7, where a local search is performed.

The aim is to provide a sequence π_λ possibly better than the initial sequence $\pi(\lambda)$, by exploring a neighbourhood of the trial solution $x(\lambda)$.

In particular, for each of the $\frac{n(n-1)}{2}$ couples of indices (i, j), with $i < j$, we check whether swapping jobs $[i]$ and $[j]$ results in the decrease of the current objective function value. If this is the case, we execute the exchange, we update the current objective function value and we iterate the process.

A remarkable consideration is that, in correspondence to any sequence π_{ij}, generated by swapping jobs $[i]$ and $[j]$ in an initial sequence π, there is no need to compute from scratch the objective function value, which would be expensive. In fact, denoting by C_A and C_B and by $C_{A_{ij}}$ and $C_{B_{ij}}$ the weighted completion times of the jobs in J_A and J_B provided, respectively, by the initial sequence π and by the generated sequence π_{ij}, it is possible to show that $C_{A_{ij}}$ and $C_{B_{ij}}$ can be determined in function of C_A and C_B, by means of the following formulae:

$$C_{A_{ij}} = C_A + \sum_{k \in J_{A_k}} w_{[k]} \left(p_{[j]} - p_{[i]}\right) + \sum_{k \in J_{A_i}} w_{[i]} p_{[k]} - \sum_{k \in J_{A_j}} w_{[j]} p_{[k]} \tag{22}$$

and

$$C_{B_{ij}} = C_B + \sum_{k \in J_{B_k}} w_{[k]} \left(p_{[j]} - p_{[i]}\right) + \sum_{k \in J_{B_i}} w_{[i]} p_{[k]} - \sum_{k \in J_{B_j}} w_{[j]} p_{[k]}, \tag{23}$$

where

$$J_{A_k} \triangleq \begin{cases} \{k \mid i < k < j\} & \text{if } [k] \in J_A \\ \emptyset & \text{otherwise,} \end{cases}$$

$$J_{A_i} \triangleq \begin{cases} \{k \mid i < k \leq j\} & \text{if } [i] \in J_A \\ \emptyset & \text{otherwise,} \end{cases}$$

$$J_{A_j} \triangleq \begin{cases} \{k \mid i \leq k < j\} & \text{if } [j] \in J_A \\ \emptyset & \text{otherwise} \end{cases}$$

and

$$J_{B_k} \triangleq \begin{cases} \{k \mid i < k < j\} & \text{if } [k] \in J_B \\ \emptyset & \text{otherwise,} \end{cases}$$

$$J_{B_i} \triangleq \begin{cases} \{k \mid i < k \leq j\} & \text{if } [i] \in J_B \\ \emptyset & \text{otherwise,} \end{cases}$$

$$J_{B_j} \triangleq \begin{cases} \{k \mid i \leq k < j\} & \text{if } [j] \in J_B \\ \emptyset & \text{otherwise.} \end{cases}$$

Although using (22) and (23) would however require a linear time in n, it is worth noting that, since such formulae involve only the jobs placed between positions i and j, avoiding the recalculation of the objective function from the beginning leads to a relevant improvement on the performance of the overall algorithm.

5. Numerical results

Our Lagrangian heuristics (Algorithm 1) has been implemented in Java (version 14.02) and tested on a Windows 10 system, characterized by 16 GB of RAM and a 2.30 GHz Intel Core i7 processor.

We observe that, in the mixed integer linear model (9), the parameter d_t provided by the Glover linearization can be interpreted as a sort of big M, since it constitutes an upper bound for the variable z_{jt}. Based on this consideration, in all the numerical experiments we have used a more stringent value, say $\bar{d}_t$, for this parameter, letting

$$\bar{d}_1 \triangleq 0$$

and

$$\bar{d}_t \triangleq \sum_{l=1}^{t-1} p_{(l)}, \quad t = 2, \ldots, n,$$

with $p_{(j)}$, for $j = 1, \ldots, n$, being the processing times such that

$$p_{(1)} \geq p_{(2)} \geq \cdots \geq p_{(n)}.$$

This choice does not affect the equivalence between the MILP formulation and the nonlinear problem (8) (see Theorem 2), since, taking into account the assignment constraints (7b)-(7d), it holds

$$\sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \leq \bar{d}_t, \quad t = 1, \ldots, n.$$

For each run, we have fixed a time limit equal to 3600 seconds and the maximum number k_{max} of iterations equal to 1000. The code stops also when an incumbent equal to zero is generated, which, due to the presence of the absolute value in the objective function, corresponds to an optimal solution of the problem providing a perfect balancing between J_A and J_B. As for the initial setting of λ, we have chosen $\lambda = e$, where e is the vector of ones.

To solve the linear assignment problem $LR_x(\lambda)$, we have used the Gurobi Optimizer software, version 9.1. Regarding the choice of the initial sequence $\bar{\pi}_{init}$, we have set it as the sequence obtained by alternating the jobs between the two classes until one class is completely scheduled, and by accommodating at the end the remaining jobs of the other class, if any.

The code has been first tested on 21 test problems listed in Table 1 and obtained for different values of n_A and n_B, up to 500 jobs. The processing times and the weights have been randomly generated from a uniform distribution on the interval $[1, 25]$ for problems 1-12 (small instances), on the interval $[1, 50]$ for problems 13-16 (medium instances) and on the interval $[1, 100]$ for problems 17-21 (large instances).

We compare our results with those ones obtained by Gurobi Optimizer in solving exactly the mixed integer quadratically constrained program (8) and the mixed integer linear program (9) (columns MIQCP and MILP of Table 1, respectively), with d_t substituted by $\bar{d}_t$ in the latter. For both programs, we have fixed as well a time limit equal to 3600 seconds and, in order to force Gurobi to generate better feasible solutions, we have set the parameters MIPFocus and Heuristics equal to 1.

In Table 1, for each solver, we report the execution time and the best objective function value found by the algorithm (the incumbent). The character "-" means that no feasible solution has been found within the time limit, while "OOM" indicates an out-of-memory failure.

In solving each problem, our code has exited with the best incumbent being equal to zero, which is a sign of optimality (perfect balancing between J_A and J_B). We remind, in fact, that zero is as well a trivial lower bound for the original problem because of the absolute value in the objective function.

Note also that our approach is very fast, whereas solving exactly the mixed integer programs appears to be prohibitive from the computational point of view: in the quadratically constrained case (column MIQCP), in most of the instances Gurobi Optimizer is not able to provide a feasible solution or fails because of the out-of-memory event, while in the linear case (column MILP) an optimal solution is found prevalently on the small instances, whereas for the medium and the large ones the out-of-memory event occurs.

A trivial greedy solution to problem (7) is that one obtainable by alternatively scheduling the jobs of the two classes and used to initialize $\bar{\pi}_{init}$. Then, in order to get an idea of how far this trivial solution is from the optimal one, we have added in Table 1 the values of the corresponding incumbent (column Greedy Incumbent), which clearly testify that the greedy approach provides a very imbalanced solution.

Table 1
Comparison of the Lagrangian heuristics against Gurobi Optimizer.

#	n	n_A	n_B	Greedy	Lagrangian heuristics		MIQCP (Gurobi)		MILP (Gurobi)	
				Incumbent	Time	Incumbent	Time	Incumbent	Time	Incumbent
1	20		10	525.50	0.021	0	139.73	0	0.940	0
2	30	10	20	1194.75	0.017	0	3600	0.50	3.05	0
3	40		30	1446.23	0.031	0	3600	2805.10	7.55	0
4	40		20	836.65	0.019	0	3600	0.350	14.20	0
5	50	20	30	2156.72	0.031	0	3600	6227.03	42.58	0
6	60		40	2049.95	0.123	0	3600	-	41.85	0
7	60		30	585.77	0.049	0	3600	-	61.46	0
8	70	30	40	2488.88	0.069	0	3600	-	128.76	0
9	80		50	1537.95	0.242	0	OOM		361.27	0
10	80		40	279.83	0.337	0	OOM		145.09	0
11	90	40	50	342.63	1.69	0	OOM		250.51	0
12	100		60	2922.91	0.288	0	OOM		528.43	0
13	100	50	50	5203.38	0.067	0	OOM		532.79	0
14	150		100	31312.22	1.02	0	OOM		3600	376.26
15	200	100	100	4066.43	3.26	0	OOM		OOM	
16	250		150	25109.42	1.49	0	OOM		OOM	
17	300	150	150	5520.81	2.97	0	OOM		OOM	
18	350		200	82117.30	97.47	0	OOM		OOM	
19	400	200	200	67275.48	146.49	0	OOM		OOM	
20	450		250	172938.22	41.95	0	OOM		OOM	
21	500	250	250	39367.53	128.12	0	OOM		OOM	

Table 2
Lagrangian heuristics on small test problems ($p_j, w_j \in [1, 25]$).

n	n_A	n_B	Time (secs)	Incumbent	# Iter	Time per iter (secs)	# Solved to opt
20		10	0.041	0	16.38	0.003	50/50
30	10	20	0.039	0	7.26	0.005	50/50
40		30	0.041	0	4.44	0.009	50/50
40		20	0.062	0	6.52	0.010	50/50
50	20	30	0.158	0	9.74	0.016	50/50
60		40	0.092	0	4.01	0.023	50/50
60		30	0.079	0	3.50	0.023	50/50
70	30	40	0.248	0	7.18	0.035	50/50
80		50	0.346	0	7.08	0.049	50/50
80		40	0.122	0	2.32	0.053	50/50
90	40	50	0.651	0	8.96	0.073	50/50
100		60	0.261	0	2.96	0.088	50/50

Table 3
Lagrangian heuristics on medium test problems ($p_j, w_j \in [1, 50]$).

n	n_A	n_B	Time (secs)	Incumbent	# Iter	Time per iter (secs)	# Solved to opt
100	50	50	0.514	0	5.54	0.093	50/50
150		100	1.04	0	3.48	0.300	50/50
200	100	100	1.89	0	2.54	0.744	50/50
250		150	8.35	0	4.82	1.73	50/50

Table 4
Lagrangian heuristics on large test problems ($p_j, w_j \in [1, 100]$).

n	n_A	n_B	Time (secs)	Incumbent	# Iter	Time per iter (secs)	# Solved to opt
300	150	150	20.99	0	5.14	4.08	50/50
350		200	65.91	0	10.50	6.28	50/50
400	200	200	66.10	0	3.38	19.56	50/50
450		250	212.55	0	8.56	24.83	50/50
500	250	250	123.29	0	3.64	33.87	50/50

To assess the Lagrangian heuristics results of Table 1, we have performed further experiments by testing our code on three additional groups of problems (small, medium and large), considering different scenarios by varying of n_A and n_B chosen in the same way as in Table 1 (see Tables 2-4). For each scenario a set of 50 instances has been generated, adopting the same above strategy in order to randomly obtain the processing times and the weights.

In Tables 2-4, for each scenario we report the following results:

- the average execution time (seconds);
- the average value of the best incumbent;
- the average number of iterations needed to compute the best incumbent;

Table 5
Lagrangian heuristics on test problems with large ranges ($p_j, w_j \in [1, 3n]$).

n	n_A	n_B	Time (secs)	Incumbent	# Iter	Time per iter (secs)	# Solved to opt
60	30	30	1.08	0	95.45	0.011	20/20
100	50	50	12.01	0	124.10	0.097	20/20
200	100	100	144.58	0	178.10	0.812	20/20
300	150	150	737.13	0	177.00	4.17	20/20
400	200	200	1467.99	0.001	138.05	10.63	16/20
500	250	250	1847.42	0.005	81.70	22.61	4/20

- the average execution time (seconds) per iteration.
- the number of instances certainly solved to optimality, i.e. when the generated incumbent is equal to zero (in the other cases the optimal objective function value is unknown and we are not able to verify the optimality).

The results reported in Tables 2-4 are coherent with those ones reported in Table 1, since in solving each problem the code has exited very fast with the best incumbent being again equal to zero.

To make the numerical experiments more compelling, we have run the code also on some additional instances characterized by larger values of p_j and w_j, which have been taken randomly as integer numbers from a uniform distribution on the interval $[1, 3n]$ (see Table 5). In such case we have considered six scenarios, and for each scenario we have generated 20 instances.

Differently from the previous type of experiments, the code did not exit always with the best incumbent equal to zero, within the fixed time limit. Nevertheless, on the other hand, it is possible to observe that in all the cases an incumbent less than or equal to 0.005 has been found on average.

From the overall results reported in Tables 2-5, it is evident how the average time per iteration is monotonically increasing with respect to the number of jobs. This behaviour is not surprising, since solving the assignment problem and performing the local search require clearly more computational effort for large values of n.

It is worth noting that, even if having $C^* = 0$ is very frequent in the practical cases (as shown by our randomly generated instances), on the other hand this is not always guaranteed. Then, for the sake of completeness, in order to analyze the behaviour of the Lagrangian heuristics in solving a non-perfectly balanced problem (i.e. a problem with $C^* > 0$), we have constructed a simple toy example characterized by 10 jobs and $C^* = 17$, whose data are listed in Table 6.

The results obtained by the Lagrangian heuristics on the toy problem are reported in Table 7, in comparison with those ones provided by Gurobi Optimizer in solving the two mixed integer problems, the quadratically constrained program (MICQP) and the linear one (MILP). Our algorithm has exited for reaching the maximum number k_{max} of iterations, performed in 0.758 seconds. On the other hand Gurobi Optimizer has solved

Table 6
Toy problem with $n_A = n_B = 5$ and $C^* = 17$.

J_A		J_B	
p_j	w_j	p_j	w_j
5	5	7	10
3	3	3	11
1	4	9	16
6	2	2	15
5	1	4	9

Table 7
Comparison of the Lagrangian heuristics against Gurobi Optimizer on the toy problem described in Table 6.

Lagrangian heuristics		MIQCP (Gurobi)		MILP (Gurobi)	
Time	Incumbent	Time	Incumbent	Time	Incumbent
0.758	17.00	91.24	17.00	20.15	17.000

the quadratically constrained program in 91.24 seconds, while in the linear case it has determined the optimal solution in 20.15 seconds.

To confirm the results obtained on the above toy problem, we finally report in Table 8 the results obtained by the Lagrangian heuristics and Gurobi Optimizer on additional 20 test problems, characterized by unweighted identical jobs (i.e. with $p_j = p_{const}$ and $w_j = 1$, for all $j \in J$) and n_A and n_B odd numbers. These problems, for which the optimal objective function value is computable a priori as $C^* = \frac{np_{const}}{2n_An_B} > 0$ (see [7]), have been obtained by randomly generating the constant p_{const} (fifth column of Table 8) from a uniform distribution on the same intervals used for generating the instances of Table 1, i.e. $[1, 25]$ for problems 1-12 (small instances), $[1, 50]$ for problems 13-16 (medium instances) and $[1, 100]$ for problems 17-21 (large instances). In all such cases, the Lagrangian heuristics code has been able to compute an optimal solution, exiting either for the maximum number of iterations or for reaching the time limit.

About the comparison against Gurobi, we can confirm the comments done on the results of Table 1, even if, in solving the MILP model, Gurobi is faster than the Lagrangian heuristics on some of the small instances. Also for such kind of problems, the greedy approach, for which in these cases the objective function value is easily computable in a closed form,[1] provides an objective function value that is always far from the optimal one.

[1] $|\bar{C_B} - \bar{C_A}| = \frac{p_{const}\left(\max\{n_A, n_B\}^2 - \min\{n_A, n_B\}^2 + n_A + n_B\right)}{2\max\{n_A, n_B\}}$.

Table 8
Comparison of the Lagrangian heuristics against Gurobi Optimizer on test problems with unweighted identical jobs and both n_A and n_B odd.

#	n	n_A	n_B	p_{const}	Greedy	Lagrangian heuristics		MIQCP (Gurobi)		MILP (Gurobi)	
					Incumbent	Time	Incumbent	Time	Incumbent	Time	Incumbent
1	30		15	11	11.00	3.05	0.733	3600	0.733	0.68	0.733
2	40	15	25	9	79.20	8.16	0.480	3600	180.00	2.22	0.480
3	50		35	10	150.00	9.17	0.476	3600	250.00	5.71	0.476
4	50		25	13	13.00	10.45	0.520	3600	325.00	5.81	0.520
5	60	25	35	13	122.57	19.75	0.446	3600	-	14.20	0.446
6	70		45	12	196.00	42.31	0.373		OOM	29.88	0.373
7	70		35	12	12.00	21.15	0.343		OOM	3600	0.343
8	80	35	45	15	146.67	31.47	0.381		OOM	56.28	0.381
9	90		55	15	257.73	44.87	0.351		OOM	102.37	0.351
10	90		45	14	14.00	43.33	0.311		OOM	102.76	0.311
11	100	45	55	14	140.00	60.28	0.283		OOM	3600	0.283
12	110		65	17	302.08	76.23	0.319		OOM	3600	0.319
13	150	75	75	43	43.00	211.41	0.573		OOM	3600	517.72
14	200		125	16	652.80	733.44	0.171		OOM		OOM
15	250	125	125	42	42.00	1927.55	0.336		OOM		OOM
16	300		175	2	87.43	3378.99	0.014		OOM		OOM
17	350	175	175	77	77.00	3600	0.440		OOM		OOM
18	400		225	1	45.33	3600	0.005		OOM		OOM
19	450	225	225	26	26.00	3600	0.116		OOM		OOM
20	500		275	54	2503.64	3600	0.218		OOM		OOM

6. Conclusions

We have introduced a new scheduling problem whose objective is to balance the average weighted completion times of two classes of jobs. For such a problem we have shown the NP-hardness and we have presented a heuristic solution approach.
The proposed method is based on the Lagrangian relaxation of a mixed integer linear model, obtained by applying the Glover linearization to a nonsmooth quadratic assignment type program. The numerical results obtained on more than one thousand randomly generated test problems have shown the efficiency and the effectiveness of our approach that has been able to compute a certified optimal solution in almost all the cases.

Future research could be oriented into two different directions. On one hand, it would be interesting to extend such problem to the case with more than two classes of jobs and, on the other hand, some research could be devoted to tackling a bi-objective version of the problem, where, in addition to the balancing, the minimization of both average weighted completion times is also performed.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] E. Aarts, J. Lenstra, Local Search in Combinatorial Optimization, John Wiley & Sons, Inc., USA, 1997.
[2] A. Agnetis, J.-C. Billaut, S. Gawiejnowicz, D. Pacciarelli, A. Soukhal, Multiagent Scheduling: Models and Algorithms, Springer, 2014.
[3] A. Agnetis, B. Chen, G. Nicosia, A. Pacifici, Price of fairness in two-agent single-machine scheduling problems, Eur. J. Oper. Res. 276 (2019) 79–87.
[4] A. Agnetis, G. De Pascale, D. Pacciarelli, A lagrangian approach to single-machine scheduling problems with two competing agents, J. Sched. 12 (2009) 401–415.
[5] A. Agnetis, P.B. Mirchandani, D. Pacciarelli, A. Pacifici, Scheduling problems with two competing agents, Oper. Res. 52 (2004) 229–242.
[6] A. Agnetis, D. Pacciarelli, A. Pacifici, Multi-agent single machine scheduling, Ann. Oper. Res. 150 (2007) 3–15.
[7] M. Avolio, A. Fuduli, A subset-sum type formulation of a two-agent single-machine scheduling problem, Inf. Process. Lett. 155 (2020) 105886.
[8] E. Bahel, C. Trudeau, Stability and fairness in the job scheduling problem, Games Econ. Behav. 117 (2019) 1–14.
[9] K. Baker, J. Smith, A multiple-criterion model for machine scheduling, J. Sched. 6 (2003) 7–16.
[10] D. Bertsimas, V.F. Farias, N. Trichakis, The price of fairness, Oper. Res. 59 (2011) 17–31.
[11] S. Brams, A.D. Taylor, Fair division - from cake-cutting to dispute resolution, 1996.
[12] R. Burkard, E. Çela, P. Pardalos, L. Pitsoulis, The quadratic assignment problem, in: D. Du, P. Pardalos (Eds.), Handbook of Combinatorial Optimization, Springer US, Boston, MA, 1998, pp. 241–337.
[13] D. Elvikis, V. T'kindt, Two-agent scheduling on uniform parallel machines with min-max criteria, Ann. Oper. Res. 213 (2014) 79–94.
[14] A. Frangioni, About Lagrangian methods in integer optimization, Ann. Oper. Res. 139 (2005) 163–193.

[15] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.
[16] M. Gaudioso, A view of Lagrangian relaxation and its applications, in: Numerical Nonsmooth Optimization: State of the Art Algorithms, 2020, pp. 579–617.
[17] M. Gaudioso, G. Giallombardo, G. Miglionico, Essentials of numerical nonsmooth optimization, 4OR 18 (2020) 1–47.
[18] F. Glover, Improved linear integer programming formulations of nonlinear integer problems, Manag. Sci. 22 (1975) 455–460.
[19] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G.R. Kan, Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey, Ann. Discrete Math., vol. 5, 1979.
[20] M. Guignard, Lagrangian relaxation, Top 11 (2003) 151–200.
[21] L. Kaufman, F. Broeckx, An algorithm for the quadratic assignment problem using Bender's decomposition, Eur. J. Oper. Res. 2 (1978) 207–211.
[22] W.-C. Lee, J.-Y. Wang, M.-C. Lin, A branch-and-bound algorithm for minimizing the total weighted completion time on parallel identical machines with two competing agents, Knowl.-Based Syst. 105 (2016) 68–82.
[23] P. Liu, M. Gu, G. Li, Two-agent scheduling on a single machine with release dates, Comput. Oper. Res. 111 (2019) 35–42.
[24] B. Moseley, S. Vardi, The efficiency-fairness balance of round robin scheduling, Oper. Res. Lett. 50 (2022) 20–27.
[25] Q. Nong, T. Cheng, C. Ng, Two-agent scheduling to minimize the total cost, Eur. J. Oper. Res. 215 (2011) 39–44.
[26] B. Polyak, Introduction to Optimization, Optimization Software Inc., New York, 1987.
[27] N. Shor, Minimizations Methods for Nondifferentiable Functions, Springer-Verlag, Berlin, 1985.
[28] Y. Yin, W.-H. Wu, S.-R. Cheng, C.-C. Wu, An investigation on a two-agent single-machine scheduling problem with unequal release dates, Comput. Oper. Res. 39 (2012) 3062–3073.
[29] F. Yu, P. Wen, S. Yi, A multi-agent scheduling problem for two identical parallel machines to minimize total tardiness time and makespan, Adv. Mech. Eng. 10 (2018).
[30] Y. Zhang, Z. Zhang, Z. Liu, The price of fairness for a two-agent scheduling game minimizing total completion time, J. Comb. Optim. (2020), https://doi.org/10.1007/s10878-020-00581-5, in press.