THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Pour l'obtention du titre de

Docteur en Informatique

École doctorale Informatique, Télécommunications et Electronique (Paris)

Ordonnancement avec dates de livraison et gains cumulatifs

présentée par Yasmina SEDDIK

JURY

M. Jacques Carlier *Professeur à l'Université de Technologie de Compiègne*	Rapporteur
M. Philippe Chrétienne *Professeur à l'Université Pierre et Marie Curie*	Examinateur
M. Stéphane Dauzère-Pérès *Professeur à l'Ecole des Mines de Saint-Etienne*	Rapporteur
M. Federico Della Croce *Professeur au Politecnico di Torino, Italie*	Examinateur
M. Christophe Gonzales *Professeur à l'Université Pierre et Marie Curie*	Directeur de Thèse
Mme. Safia Kedad-Sidhoum *Maître de Conférence HDR à l'Université Pierre et Marie Curie*	Directrice de Thèse
M. Pascal Wirth *Président Directeur Général de Banctec France*	Invité

Remerciements

Premièrement, je tiens à exprimer toute ma gratitude à mes directeurs de thèse, Safia Kedad-Sidhoum et Christophe Gonzales, qui m'ont inlassablement épaulée tout au long de cette thèse, en rendant cette expérience enrichissante et vraiment très agréable. Ils ont su conjuguer leurs qualités respectives pour me faire bénéficier d'un encadrement à la fois rigoureux et enthousiaste, dans la bonne humeur et dans l'écoute. Je ne peux que leur dire un grand merci pour m'avoir si bien guidée pendant ces trois ans!

Je remercie sincèrement Jacques Carlier et Stéphane Dauzère-Pérès, pour avoir accepté de rapporter ce travail, et pour leur lecture attentive de la thèse et les remarques constructives qu'ils m'ont faites.

Je remercie les examinateurs d'avoir accepté de participer à ce jury : Federico Della Croce, dont j'espère qu'il aura apprécié ces travaux; et Philippe Chrétienne, dont les précieux conseils dispensés lors de séminaires d'équipe ont contribué à améliorer ces travaux.

Je remercie également Pascal Wirth, président directeur général de Banctec, pour sa participation au jury. Son soutien a permis d'initier une collaboration avec le Lip6 dans le cadre du projet Dem@tFactory qui a permis la définition du périmètre de ces travaux de recherche.

Mes remerciements vont également aux financeurs du projet Dem@tFactory, grâce auquel deux ans de thèse ont pu être financés.

Je tiens également à remercier deux personnes qui ont directement contribué à ce manuscrit : Luciana pour notre agréable collaboration qui a produit un des chapitres de cette thèse, ainsi que Fanny, pour avoir vaillamment relu la preuve d'approximation.

Je remercie Olivier pour m'avoir maintes fois écoutée et conseillée sur divers aspects de l'enseignement et de la recherche.

J'adresse mes remerciements chaleureux à tous les membres des équipes Décision et RO, pour m'avoir accueillie avec cette convivialité qui rend le travail quotidien si agréable. J'ai trouvé au sein de ces équipes une ambiance chaleureuse, des collègues toujours prêts à m'aider et me conseiller, ainsi que des personnes d'une grande sympathie. Merci!

Je tiens aussi à remercier les nombreux autres collègues cotôyés au long de ces trois

ans, au laboratoire, en enseignement, en conférence..., qui m'ont témoigné leur bienveillance et leur sympathie.

Enfin, je remercie ma famille et mes amis, certains lointains géographiquement mais tous proches dans leur affection; en particulier merci à mes parents sans lesquels je n'en serais pas arrivée là, et bien sûr à Nicolas, pour son inlassable soutien au quotidien.

Table of Contents

List of Figures

List of Tables

List of Symbols

$B(S_k)$	The starting time of the first job of S_k, page 33
$b_l(S)$	The starting time of job J_l in schedule S, page 33
$C_i(S)$	Completion time of job J_i in schedule S, page 8
$C_{\max}(S_k)$	The completion time of the last job of S_k, page 33
$C_{il}(S)$	The completion time of job J_i on machine M_l in schedule S, for the flowshop problem, page 111
$\overline{C}^1_{max}$	The completion time of the last $\overline{S}^{D_1}$-job in S'_1, page 70
D_0	Time 0, page 6
$D_1, \ldots, D_K$	Delivery dates of an instance, page 6
D_{K+1}	Time before which all jobs can complete, page 6
D'_k	The alternative delivery date corresponding to D_k, for the computation of an upper bound in the Branch and Bound, page 45
D	Unique delivery date of the Single Delivery Date problem, page 27
$\Delta(S^a_k)$	The value of the maximal possible right-shift of S^a_k, page 44
$\delta(S^a_{k+1})$	The length of the right-shift of S^a_{k+1}, in order to compute U_k, page 47
$\prec_{ERD}$	Earliest Release Date order, page 6
Γ	A sequence of jobs
Γ^m	The sequence $J_{i_m}, \ldots, J_{i_1}$, in SDD-algorithm, page 29

Γ_{i_m}	Sequence constructed by SDD-algorithm at iteration m, page 29
I_k	Interval $]D_{k-1}, D_k]$, $k = 1, \ldots, K+1\}$, page 7
J_i	i-th job of an instance, $i = 1, \ldots, N$, page 6
$\mathcal{J}^{all}$	Set of jobs of an instance, page 6
$\mathcal{J}$	A set of jobs
$\mathcal{J}(\Gamma)$	The set of the jobs of sequence Γ, page 21
$\mathcal{J}(S)$	The set of the jobs of (partial) schedule S, page 21
K	Number of delivery dates of an instance, page 6
L_k	$C_{max}(S_k^a) - \min(B(S_k^a))$, page 44
M_l	Machine l, for the flowshop problem, page 111
N	Number of jobs of an instance, page 6
n^a	The node, in the Branch and Bound tree, associated to (partial) schedule S^a, page 36
$\overline{n}_1$	The number of $\overline{S}^{D_1}$-jobs in S'_1, in DS-algorithm, page 73
n_2	The number of S^{D_1}-jobs in S'_2, in DS-algorithm, page 73
n_3	The number of S^{D_1}-jobs in S'_3, in DS-algorithm, page 73
p_i	Processing time of job J_i, page 6
$p(\Gamma)$	The total processing time of the jobs of sequence Γ, page 21
$p(\mathcal{J})$	The sum of the processing times of the jobs of set $\mathcal{J}$, page 21
$p(S)$	The total processing time of the jobs of (partial) schedule S, page 21
r_i	Release date of job J_i, page 6
r'_i	The release date on the shrunk horizon, corresponding to r_i, for the computation of an upper bound in the Branch and Bound, page 47
$\tilde{r}_i$	The release date equivalent to r_i, when we only consider scheduling jobs on the employable intervals, for the computation of an upper bound in the Branch and Bound, page 46
S	A schedule

σ	A schedule
S_k	The subschedule of S containing the jobs that complete into the interval I_k, page 33
$S_i.S_j$	The concatenation of subschedule S_i with subschedule S_j, page 33
S^a	The (partial) schedule associated to node n^a of the Branch and Bound tree, page 36
S^{D_k}	The schedule obtained with $SDD(\mathcal{J}^{all}, 0, D_k)$, page 42
$sched(\Gamma)$	The schedule that schedules the jobs of Γ, in the order given by Γ, without idle times and with the last job completing at I_{up} (input of SDD_I), page 28
$u\_bound(S^a)$	An upper bound on the payoffs of the schedules that can be obtained by completing the partial schedule S^a, page 40
U_k	An upper bound on the number of jobs that can complete at or before D_k, page 42
$U_k(S^a)$	An upper bound on the number of jobs that can be inserted into S^a, while completing at or before D_k, page 43
$v(S)$	Payoff of schedule S, page 8
$V_k(S)$	The number of jobs completing at or before D_k in schedule S, page 7
$v_S(J_i)$	The payoff earned by job J_i in schedule S, page 8

Introduction

As the web becomes increasingly the place where to share knowledge, the digitization of books and other paper documents is an issue that becomes more and more important. In France, all the published books are acquired and stored by the Bibliothèque Nationale de France (BNF), since 1537, when the king of France François I enjoined the printers and librarians to submit to the library every printed book available for sale in the kingdom. Nowadays, the BNF collection includes about 14 millions of printed works and 12 millions of engravings, amongst others. Beyond the wish to share knowledge, there is also the will to preserve the more fragile and ancient documents. For these purposes, the BNF has started to digitize the items of its collection since 1990. At present, almost 2 million books have been digitized. Among them, 100 000 per year were digitized between 2008 and 2010, thanks to the mass digitization program Dem@tFactory launched by the BNF. Seven partners were involved in this FUI project: the industrial partners Safig (leader of the project), A2IA, Banctec and Temis; and three research laboratories: Cedric (CNAM), A2SI (ESIEE) and LIP6 (UPMC). This thesis is part of Dem@tFactory project, concerning the optimization of the workflow which includes several tasks for each digitized book. However, since we could not yet get accurate informations on the practical workflow problem, we worked on a theoretical problem that takes into account part of the specifications of the digitization issue. The direct collaborator of LIP6 in the project was Banctec, a digitization firm, with which we worked to obtain the specifications of the problem. However, in April 2010, the leader of the project, Safig, has been taken over by another firm, Jouve, that became by then the leader of the project. Nonetheless, during the several months that this take over process lasted, the project accumulated delays. As a consequence, not all the informations could be retrieved from Banctec.

The contributions of this work are the following. We modeled the digitization scheduling problem as a single machine scheduling problem with a new criterion: cumulative payoffs depending on common delivery dates. We established the strong NP-hardness of the problem in the general case, and identified a weakly NP-hard special

case (2DD) and four polynomial cases. For one of these polynomial cases, we designed an algorithm (SDD-algorithm) on which rely many of the further results. We designed and implemented exact resolution methods, both for the general problem (Branch and Bound) and for 2DD (dynamic programming). For these two methods, bounds and dominance rules specific to the problem were identified. Moreover, for 2DD, we provided a polynomial algorithm with an absolute guarantee of 1. Finally, in order to consider a model that is closer to the real world problem, we studied, as a joint work with post-doctoral researcher Luciana Pessoa, a permutation flowshop problem with cumulative payoffs depending on common delivery dates, for which we implemented constructive heuristics, local search methods and a metaheuristic.

This document is organized as follows. In Chapter 1, we present the real world problem of book digitization, its modeling as a single machine scheduling problem and the state of the art about similar problems. In Chapter 2, complexity results are established: the strongly NP-hardness of the general problem, the NP-hardness of 2DD and four polynomial cases. In Chapter 3, a Branch and Bound method is presented, in order to solve the general problem. Chapter 4 is dedicated to solving the 2DD problem. First, a dynamic programming algorithm is presented, which establishes the weakly NP-hardness of the problem; some experimental results on this method are presented. Then, a polynomial algorithm with a performance guarantee is described. In Chapter 5, we consider the permutation flowshop problem with cumulative payoffs depending on common delivery dates. Finally, we draw some conclusions and research issues.

Chapter 1

The digitization scheduling problem

In this chapter, we first present the Banctec digitization workflow, pointing out its main constraints and requirements. Then, starting from these specifications, we establish the model of the corresponding scheduling problem. Finally, we present a state of the art on similar scheduling problems.

1.1 The digitization's planning issue

We consider here the point of view of the manufacturer, Banctec, whose workflow is depicted in Figure 1.1. It can be seen that the process is mainly linear and is constituted of four main steps: Digitization, First quality control, Segmentation, Second quality control (only the Table of contents creation is made in parallel with the Segmentation). A huge number of works must be digitized following this linear process.

Moreover, each book has its own features. For instance, the degree of fragility of a book determines the kind of digitization techniques and resources that can be used. Hence, a more fragile book will be longer to digitize. Another example of a feature is the belonging to a collection, since the digitization processes of the books of a same collection are not independent. Indeed, all the books of a collection have a common table of contents, hence the digitization process of the books of a collection only completes when all those books have been processed.

This outline brings out the usefulness of an automatic tool that helps the manufacturer in planifying the digitization of the books.

As said above, although the BNF is the client, we collaborated with the manufacturer, i.e. the digitization firm Banctec. Hence, in this thesis, we focus on the manufacturer's point of view, by trying to meet the client requirements while maximizing the manufacturer's own profits. Unfortunately, due to the already mentioned problems of Safig, the project has accumulated some delays, and therefore not all the features of the problem could be precisely retrieved, nor were any representative data of real digitization situa-

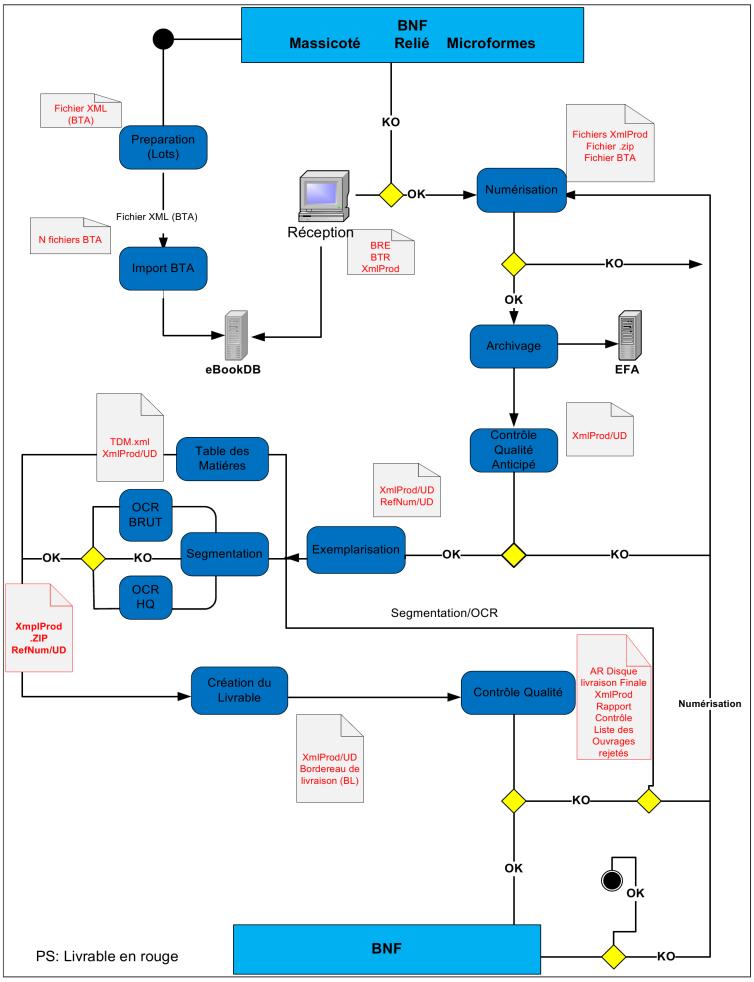


Figure 1.1: The digitization process (in french).

tions. We therefore focus on a theoretical scheduling problem, taking into account the features of the digitization's planning issue that could be ascertained.

Although the problem can be satisfactorily modeled as a flowshop problem, and as this is a first work on this problem, we mostly focus in this thesis on a single machine problem (in order to concentrate on the new optimization criterion). However, we investigate in Chapter 5 a flowshop problem.

Since each book has different characteristics (for instance the fragility, the number of pages, the presence of images, the required segmentation quality: normal or high), we consider that each digitization job has a different duration. In the scheduling problem, this corresponds to a specific processing time for each job. Moreover, the books to be digitized become available to the manufacturer at different dates: at regular intervals (once a week), the manufacturer receives parcels of books. For each parcel, its content

and date of arrival are previously known by the manufacturer. Hence, release dates have to be considered in the scheduling problem.

Finally, the client (BNF) sets several delivery dates (every four or five months) and a target quantity of digitized books for each of them. If the target quantities are not reached, the manufacturer incurs penalty costs, proportional to the number of missing items. However, when we retrieved these informations, the penalties were not yet applied, since it would have been too disadvantageous for the manufacturer, whose workflow was initially not conceived for such a huge quantity of documents.

More formally, let D_k, $k = 1, \ldots, K$ (such that $0 < D_1 < \cdots < D_K$) be those delivery dates, and Q_k, $k = 1, \ldots, K$ the corresponding target quantities. For a given $k = 1, \ldots, K$, Q_k is the number of books the client expects to be digitized from the very beginning until date D_k. Consider the example of Figure 1.2: the client demands that 15000 books are digitized 3 months after the beginning of the project. Moreover, 35000 books are demanded to be digitized 6 months after the beginning of the project. These 35000 books include all the books digitized from the beginning of the project. For the following delivery dates, we have the following goals: 60000 books must be digitized after 9 months, and 80000 after 1 year. A cumulative aspect can be noticed, since each goal includes the amount achieved at the previous delivery dates.

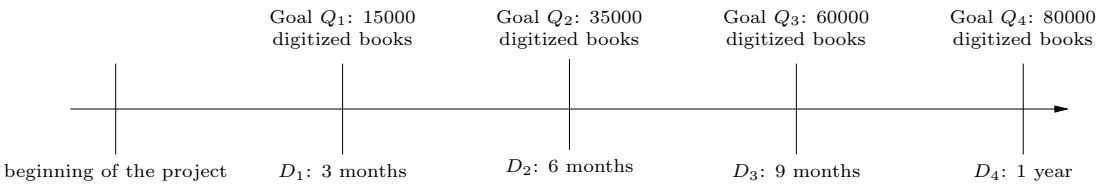


Figure 1.2: An example of delivery dates with corresponding goals set by the client.

Let V_k be the number of books having been digitized from the very beginning until date D_k, for every $k = 1, \ldots, K$. It is desirable that V_k is at least equal to Q_k if possible, or else, that V_k be as close as possible to Q_k. Let us see an example, on Figure 1.3, with the same goals as in the example of Figure 1.2. Three months after the beginning of the project, 16000 books have been digitized. Therefore, the first goal of 15000 is met. Three months later (six months after the beginning), only 30000 books have been digitized instead of 35000. Another three months later, 58000 books have been digitized instead of 60000; and a year after the beginning, 79000 instead of 80000.

A solution with maximal client satisfaction would be having $V_k \geq Q_k$ for every $k \in \{1, \ldots, K\}$. In other words, if the differences $V_k - Q_k$ for each $k = 1, \ldots, K$, are all greater than or equal to zero, the client has the maximal satisfaction. Otherwise, the smaller the difference (the greater in absolute value), the less the client's satisfaction. Besides, the manufacturer wishes to satisfy the client, and to maximize its profits (which is related to the number of digitized books). Therefore, the manufacturer prefers maximizing the number of digitized books at each delivery date, instead of simply attaining the goal

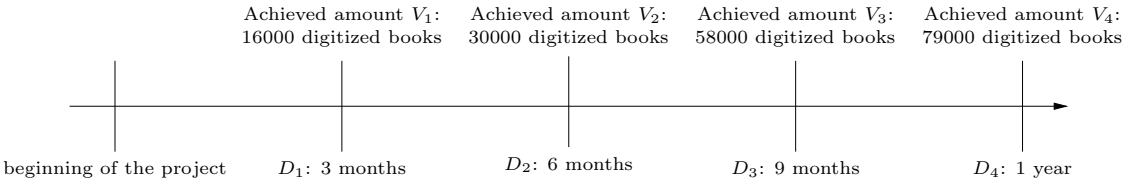


Figure 1.3: An example of achieved amounts of digitized books at each delivery date.

fixed by the client. Hence, maximizing $V_k - Q_k$ when $V_k < Q_k$ and maximizing V_k otherwise, is equivalent to simply maximize V_k, since Q_k is a constant. Overall, the wish is to maximize V_k, for each $k = 1, \ldots, K$. To aggregate the K criteria induced by the K delivery dates, we consider sum $\sum_{k=1}^{K} V_k$ to be the manufacturer's goal, which reflects its following preference: earning a given payoff is more valuable earlier than later. Indeed, summing these cumulative values gives a greater weight to the earlier completed tasks. On the example of Figure 1.3, the payoff is thus $16000+30000+58000+79000 = 183000$.

Starting from the features described above, we can define the formal problem.

1.2 The formal definition of the problem

We address here a single machine non-preemptive scheduling problem where the N jobs of the set $\mathcal{J}^{all}= \{J_1, \ldots, J_N\}$ have to be scheduled. Each job J_iJ_i has a release date $r_i \geq 0$ and a processing time $p_i > 0$.

The following total order, called ERD order (for Earliest Release Date) is defined on the jobs of $\mathcal{J}^{all}$. Given two jobs J_i and J_j, we denote by $J_i \prec_{ERD} J_j$ the assertion that J_i precedes J_j in the ERD order.

- $(r_i < r_j) \Rightarrow J_i \prec_{ERD} J_j$;
- $(r_i = r_j$ and $p_i < p_j) \Rightarrow J_i \prec_{ERD} J_j$;
- $(r_i = r_j$ and $p_i = p_j$ and $i < j) \Rightarrow J_i \prec_{ERD} J_j$;

The third condition is clearly arbitrary, in order to break ties and to ensure that ERD is a total order. This will be important to avoid the need of treating equivalent situations. On the example of Figure 1.4, we have: $J_2 \prec_{ERD} J_5 \prec_{ERD} J_3 \prec_{ERD} J_1 \prec_{ERD} J_4 \prec_{ERD} J_6$.

K delivery dates are given: $D_1, \ldots, D_K$, with $0 < D_1 < \cdots < D_K$. We also set for simplicity's sake:

- $D_0 = 0$,
- $D_{K+1} = \max(D_K, \max_{i=1,\ldots,N} r_i) + \sum_{i=1}^{N} p_i$,

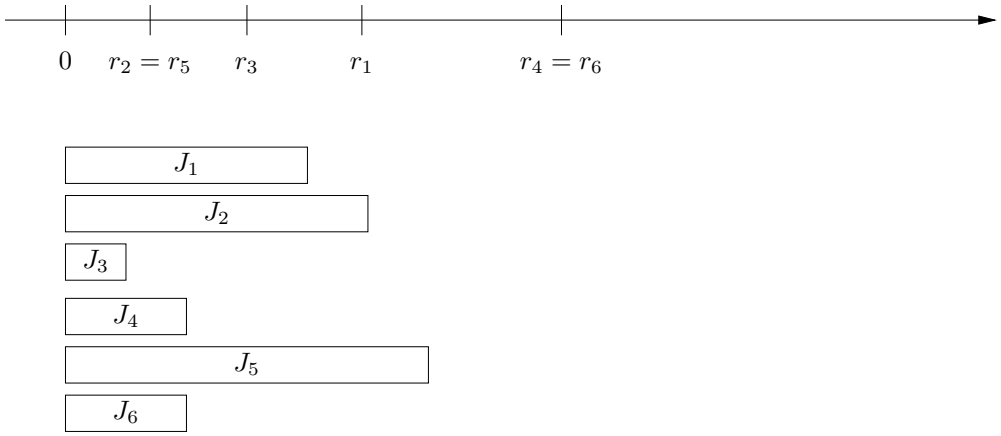


Figure 1.4: A set of jobs: on top the release dates, and at bottom the processing times.

We define $I_k =]D_{k-1}, D_k]$, for each $k \in \{1, \ldots, K+1\}$.

All the data of the problem are integer.

Figure 1.5 gives an example of delivery dates (assuming all the release dates are smaller than D_K: hence the value of D_{K+1}).

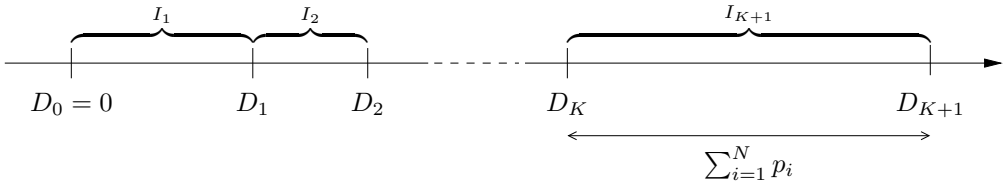


Figure 1.5: The delivery dates.

We will frequently use the following terms:

- A *schedule* is a set of completion times, one for each job of $\mathcal{J}^{all}$.
- A *partial schedule* is a set of completion times, one for each job of a given set $\mathcal{J} \subset \mathcal{J}^{all}$.
- A *subschedule* of a given schedule S is a partial schedule S' where the completion times of the jobs of S' are the same in both S and S'.
- A *block* is a (partial) schedule where there are no idle times, i.e. each job of the block (except the first) starts at the completion time of the previous job.

Given a schedule S, which is characterized by the completion times of all the jobs, $V_k(S)$ (denoted as V_k when no ambiguity is possible) represents the number of jobs that complete at or before D_k in S, $k = 1, \ldots, K$. The payoff of a given schedule S is defined

by $v(S)= \sum_{k=1}^{K} V_k(S)$. On the example of Figure 1.6, the payoff is computed as follows. Two jobs (J_1 and J_2) complete at or before D_1, therefore $V_1 = 2$. Three jobs (J_1, J_2 and J_3) complete at or before D_2, therefore $V_2 = 3$. Four jobs (J_1, J_2, J_3 and J_4) complete at or before D_3, therefore $V_3 = 4$. Overall, the total payoff is $V_1 + V_2 + V_3 = 9$.

Figure 1.6: Example of a schedule with three delivery dates.

We saw how to obtain the payoff of a schedule, by summing the payoffs corresponding to each delivery date. Let us see another way of computing the payoff, based on the jobs, instead of the delivery dates. We will refer to the payoff related to job J_i in schedule S as $v_S(J_i)$ ($v(J_i)$ when no ambiguity is possible). $v_S(J_i)$ is the number of delivery dates before or at which J_i completes in S. Let $C_i(S)$ (or C_i when no ambiguity is possible) be the completion time of job J_i in S; $v(J_i)$ can be represented by the following decreasing stepwise function:

$$v(J_i) = \begin{cases} K & \text{if } 0 < C_i \leq D_1 \\ \vdots & \\ 2 & \text{if } D_{K-2} < C_i \leq D_{K-1} \\ 1 & \text{if } D_{K-1} < C_i \leq D_K \\ 0 & \text{if } D_K < C_i \end{cases}$$

which can be more shortly described as $v(J_i) = K - k + 1$ if $C_i \in I_k$, $k = 1, \ldots, K$. This stepwise function is depicted in Figure 1.7 for $K = 6$.

Again on the example of Figure 1.6, the payoff can be computed in the following way. Both jobs J_1 and J_2 complete in $I_1 =]0, D_1]$, therefore $v(J_1) = v(J_2) = 3$. J_3 completes in $I_3 =]D_2, D_3]$, therefore $v(J_3) = 1$. J_4 completes in $I_2 =]D_1, D_2]$, therefore $v(J_4) = 2$. Overall, the total payoff is $v(J_1) + v(J_2) + v(J_3) + v(J_4) = 9$.

Both methods of computation of a schedule's payoff will be used, depending on the situation.

Extending the three-field notation of Graham et al. [11], the addressed problem can be defined as $1|r_i| \sum_{k=1}^{K} V_k$.

Finally, consider the following overall example. We are given $N = 5$ jobs J_1, J_2, J_3, J_4, J_5, such that $p_1 = 2, p_2 = 3, p_3 = 4, p_4 = 5, p_5 = 7$ and $r_1 = 22, r_2 = 8, r_3 = 3, r_4 = 13, r_5 = 18$. The ERD order on these jobs is: $J_3 \prec_{ERD} J_2 \prec_{ERD} J_4 \prec_{ERD} J_5 \prec_{ERD} J_1$.

Moreover, there are $K = 3$ delivery dates $D_1 = 10, D_2 = 16, D_3 = 27$. We set: $D_4 = D_3 + \sum_{i=1}^{5} p_i = 48$. Hence, $I_1 =]0, 10], I_2 =]10, 16], I_3 =]16, 27], I_4 =]27, 48]$.

In the feasible schedule S of Figure 1.8 $C_1(S) = 25 \in I_3$, $C_2(S) = 13 \in I_2$, $C_3(S) = 8 \in I_1$, $C_4(S) = 18 \in I_3$, $C_5(S) = 35 \in I_4$. Hence, if we compute the payoff by jobs,

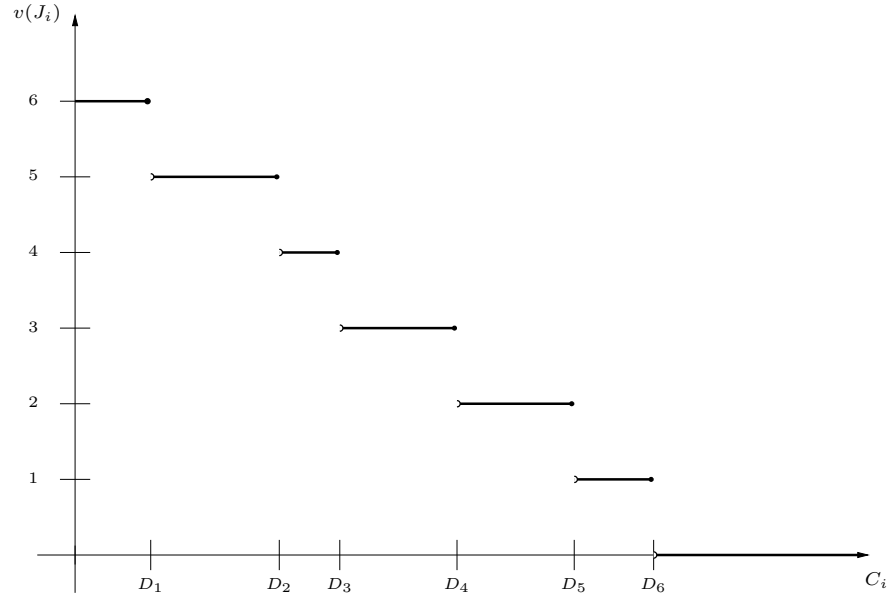


Figure 1.7: The stepwise job payoff function.

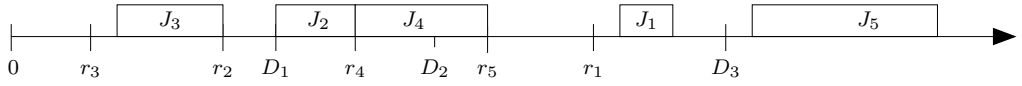


Figure 1.8: A schedule S.

we have: $v_S(J_1) = 1$, $v_S(J_2) = 2$, $v_S(J_3) = 3$, $v_S(J_4) = 1$, $v_S(J_5) = 0$. Thus, the total payoff is $v(S) = 7$. If we compute the payoff by delivery dates, $V_1(S) = 1$ since $C_3(S) \leq D_1$; $V_2(S) = 2$, since J_1 and J_2 complete before D_2; and $V_3(S) = 4$ since J_1, J_2, J_3, J_4 complete before D_3. Hence the total payoff is $v(S) = 7$.

In the following section, we make a survey of the problems related to $1|r_i|\sum_{k=1}^{K} V_k$.

1.3 State of the art

The objective function $\sum_{i=1}^{N} v(J_i)$ is a special case of the regular sum objective functions $\sum_{i=1}^{N} f_i(C_i)$, where $f_i(C_i)$ is a nondecreasing cost function or a nonincreasing payoff function. Some other special cases are, for instance, the classical criteria $\sum C_i$ (the job cost function is linear), $\sum T_i$ (the job cost function is linear, starting from its due date), $\sum U_i$ (the job cost function is stepwise, with a unique breakpoint). Raut et al. [26] consider such a general objective function, $\sum_{i=1}^{N} f_i(C_i)$, on a single machine without release dates, for which they provide several list strategy heuristics, based on the two features specific to each job: processing time and cost function.

In Section 1.3.1 we consider the works with stepwise job cost functions, while in

Section 1.3.2 we mention some other related problems.

1.3.1 Stepwise job cost functions

The objective function $\sum_{k=1}^{K} V_k$ is also a special case of the objective functions obtained as a sum of stepwise job payoff (or cost) functions, $\sum_{i=1}^{N} v(J_i)$, where each job J_i has its own stepwise payoff (or cost) function $v(J_i)$. $v(J_i)$ is characterized by its moments of value change (also called jump points or breakpoints) $D_{1,i}, \ldots, D_{K_i,i}$, and the corresponding values $w_{1,i}, \ldots, w_{K+1,i}$, as explicited below:

$$v(J_i) = \begin{cases} w_{1,i} & \text{if } 0 < C_i \leq D_{1,i} \\ w_{2,i} & \text{if } D_{1,i} < C_i \leq D_{2,i} \\ \vdots & \\ w_{K_i,i} & \text{if } D_{K_i-1,i} < C_i \leq D_{K_i,i} \\ w_{K_i+1,i} & \text{if } D_{K_i,i} < C_i \end{cases}$$

Notice that if $v(J_i)$ is a payoff (resp. cost) function, it is a decreasing (resp. increasing) stepwise function, i.e. $w_{1,i} > \cdots > w_{K_i,i}$ (resp. $w_{1,i} < \cdots < w_{K_i,i}$) and the total objective function $\sum_{i=1}^{N} v(J_i)$ has to be maximized (resp. minimized). Both problems are equivalent.

Section 1.3.1.1 is dedicated to problems without release dates, while in Section 1.3.1.2 we survey two works with release dates and stepwise cost functions.

1.3.1.1 Problems without release dates

The general stepwise objective function described above is considered by Detienne et al. [8] and Curry and Peters [4].

Detienne et al. [8] consider the single machine problem without release dates, for which they obtain very good Lagrangean bounds. These bounds are then exploited in an exact method dealing with a graph representation of the scheduling problem. Experimentations are conducted on instances with up to 500 jobs, with better results than already existing exact methods. A straightforward adaptation of the model is considered that includes release dates, but is not very efficient since it cannot solve some 30 and 50-job instances.

Curry and Peters [4] deal with an online problem on parallel machines with reassignments, where jobs arrive at regular intervals (every day) and stepwise increasing job cost functions must be minimized. When a new set of jobs arrives, the jobs that are currently being executed are not stopped, but a new optimal schedule is obtained with a Branch and Price method, including the last arrived jobs, and the jobs that were already in the system but have not yet been scheduled. Hence, at each rescheduling, all the jobs are already available, thus there is no need to deal with release dates. However,

an additional rescheduling cost is taken into account, for the jobs that had already been assigned to a machine, and are associated to a different machine in the new schedule.

Some special cases of the general stepwise objective function defined above are presented below. You can refer to Table 1.1, which classifies all the cited problems of Section 1.3.1 w.r.t. their objective function. Table 1.2 lists the NP-hard problems studied in this thesis.

Job dependent stepwise functions	**Common number of breakpoints**	**Common breakpoints**
$1\|\|\sum v(J_i)$ [8]	$1\|K_i = 1\|\sum v(J_i)^*$ [15]	$1\|D_{k,i} = D_k\|\sum v(J_i)^*$ [15, 35]
$P\|reassign\|\sum v(J_i)$ [4]	$1\|K_i = K\|\sum v(J_i)$ [15, 16, 32]	$1\|D_{k,i} = D_k, nondecr\|\sum v(J_i)^{**}$ [16]
$R\|r_i\|\sum v(J_i)^{**}$ [7]	$1\|K_i = K, nondecr\|\sum v(J_i)^{**}$ [16]	$Rm\|D_{k,i} = D_k, fixed\ K, nondecr\|\sum v(J_i)^*$ [16]
$1\|r_i\|\sum v(J_i)^{**}$ [27]	$R\|K_i = K, nondecr\|\sum v(J_i)^{**}$ [16]	

Table 1.1: The problems with objective functions that generalize $\sum V_k$, classified from the more general objective function (left) to the less general (right). All the problems in this table are NP-hard. The problems with * (resp. **) are known to be weakly (resp. strongly) NP-hard.

Common fixed stepwise function
$1\|r_i, D_{k,i} = D_k, w_{ki} = K - k + 1\|\sum v(J_i)^{**}$ (i.e. $1\|r_i\|\sum V_k$)
$1\|r_i, D_{k,i} = D_k, fixed\ K, w_{ki} = K - k + 1\|\sum v(J_i)^*$ (i.e. $1\|r_i, fixed\ K\|\sum V_k$)
$1\|r_i, D_{k,i} = D_k, K = 2, w_{ki} = K - k + 1\|\sum v(J_i)^*$ (i.e. $1\|r_i\|V_1 + V_2$)

Table 1.2: The NP-hard problems considered in this thesis.

Janiak and Krysiak [15] consider the single machine problem without release dates, with the objective function $\sum_{i=1}^{N} v(J_i)$ where all the jobs have the same number of breakpoints, i.e. $K_i = K$ for all $i \in \{1, \ldots, N\}$. This problem can be denoted as $1|K_i = K|\sum_{i=1}^{N} v(J_i)$, and is NP-hard, as its special case $1|K_i = 1|\sum_{i=1}^{N} v(J_i)$ is equivalent to $1||\sum w_i U_i$ (weakly NP-hard). Janiak and Krysiak [15] provide some list strategies heuristics for $1|K_i = K|\sum_{i=1}^{N} v(J_i)$, based on processing times p_i and weights $w_{1,i}$ of the jobs; and two strategies based on modifications of Moore-Hodgson's algorithm for $1||\sum U_i$ [21].

Moore-Hodgson's algorithm is also exploited by Tseng et al. [32] for the same problem, as a part of a constructive heuristic algorithm. Moreover, Tseng et al. [32] define some neighborhood structures and a Variable Neighborhood Search method, for which

they present experimental results for instances with up to 50 jobs.

An adaptation of the Moore-Hodgson's algorithm is also widely used in this thesis, and is presented as SDD-algorithm in Chapter 2.

The special case where the breakpoints are common to all the jobs, which is denoted as $1|D_{k,i} = D_k|\sum_{i=1}^{N} v(J_i)$, is considered by Janiak and Krysiak [15], which provide a pseudopolynomial algorithm. For the same problem, Yang [35] provides a Branch and Bound method that uses a similar branching structure as the one used in the Branch and Bound method that we present in Chapter 3, that consists in choosing the stage (the interval between two delivery dates) where to schedule a given job. Yang [35] present satisfying experimental results for instances with up to 100 jobs.

Janiak and Krysiak [16] consider the variant of the objective function where the payoff stepwise function is nondecreasing, i.e. $w_{1,i} \geq \cdots \geq w_{K_i,i}$; and where all the jobs have the same number of breakpoints: $1|K_i = K, nondecr|\sum_{i=1}^{N} v(J_i)$. They show that the single machine problem where the breakpoints are common $1|D_{k,i} = D_k, nondecr|\sum_{i=1}^{N} v(J_i)$ is already strongly NP-hard. They also provide a pseudopolynomial time algorithm for the unrelated machines problem where the number m of machines is fixed, the jobs have common breakpoints, and the number K of the breakpoints is fixed: $Rm|D_{k,i} = D_k, Kfixed, nondecr|\sum_{i=1}^{N} v(J_i)$. Finally, they propose several heuristics to solve the general problem with unrelated parallel processors $R|K_i = K, nondecr|\sum_{i=1}^{N} v(J_i)$. These heuristic methods are obtained by combining different list strategies (based on processing times p_i and weights $w_{1,i}$) to obtain an input list, with different strategies for assigning and then sequencing the jobs on the machines.

Raut et al. [26], Detienne et al. [8], Curry and Peters [4], Janiak and Krysiak [15, 16], Tseng et al. [32] and Yang [35] mainly focus on the hardness of the objective function, and do not consider release dates. Hence, despite the similarity of the objective function, the same structural properties do not apply for $1|r_i|\sum_{k=1}^{K} V_k$. Indeed, in $1|r_i|\sum_{k=1}^{K} V_k$ we have a more specific objective function, however, as shown in Chapter 2, $1|r_i|\sum_{k=1}^{K} V_k$ is still strongly NP-hard, while its relaxed version $1||\sum_{k=1}^{K} V_k$ without release dates is polynomially solvable.

1.3.1.2 Problems with release dates

To the best of our knowledge, only two works consider both stepwise job cost functions as objective, and release dates as a constraint of the problem: those of Sahin and Ahuja [27] and of Detienne et al. [7]. We give a more detailed review of these two works, as the studied problems are the more closely related to $1|r_i|\sum V_k$.

The results of Sahin and Ahuja [27] highlight that time-indexed formulations are not ideal to solve large instances of $1|r_i|\sum v(J_i)$. For this reason, Detienne et al. [7] provide dedicated Integer Linear Programming (ILP) formulations.

Sahin and Ahuja [27] consider the single machine problem with release dates and stepwise job cost functions. They propose two time-indexed formulations. The first formulation, called X-IP, is based on completion times of the jobs and is known to provide strong lower bounds for $1|r_j|\sum C_j$, but is difficult to solve even as an LP, due to its size, as the number of variables and constraints is pseudopolynomial. The X-IP formulation follows.

$$(X-IP)\begin{cases} \min \sum_{j=1}^{N}\sum_{t=1}^{T} c_{jt}x_{jt} & & (1.1)\\ \text{s.t.} \sum_{t=r_j+p_j}^{T} x_{jt} = 1, & j \in \{1,\dots,N\} & (1.2)\\ \sum_{j=1}^{N}\sum_{t'=t}^{t+p_j-1} x_{jt'} = 1 & t \in \{1,\dots,T\} & (1.3)\\ x_{jt} \in \{0,1\}, & j \in \{1,\dots,N\}, t \in \{1,\dots,T\} & (1.4) \end{cases}$$

where:

- $c_{jt} = \begin{cases} 0 & \text{if } t < D_{1,j} \\ w_{k,j} & \text{if } D_{k-1,j} < t \le D_{k,j}, k = 1,\dots,K_j - 1 \\ w_{K_j,j} & \text{if } D_{K_j,j} < t \le T \end{cases}$
- $T = \max_{j=1,\dots,N} r_j + \sum_{j=1,\dots,N} p_j$
- $x_{jt} = 1$ if job J_j completes at time t, 0 otherwise

X-IP model can be straightforwardly adapted to $1|r_i|V_k$, by replacing c_{jt} by c_t, as all the jobs have the same cost/payoff function.

The second formulation, Y-PR, is a relaxation based on the following slightly modified idea of preemption.

Each job J_j, $j = 1,\dots,N$, is represented by a set $L_j = \{J_{j,1},\dots,J_{j,p_j}\}$ of p_j unit-time tasks. Each unit-time task of L_j has the same release date as its parent job J_j. When a unit-time task $J_{j,l}$ completes at time t, its cost is c_{jt}/p_j.

$$(Y-PR)\begin{cases} \min \sum_{j=1}^{N}\sum_{l=1}^{p_j}\sum_{t=1}^{T}(c_{jt}/p_j)y_{jlt} & (1.5)\\ \text{s.t. } \sum_{t=r_j}^{T} y_{jlt}=1, \qquad j\in\{1,\dots,N\}, l\in L_j & (1.6)\\ \sum_{j=1}^{N}\sum_{k=1}^{p_j} y_{jlt}\le 1, \qquad t\in\{1,\dots,T\} & (1.7)\\ y_{jlt}\in\{0,1\}, \qquad j\in\{1,\dots,N\}, l\in L_j, t\in\{1,\dots,T\} & (1.8) \end{cases}$$

where:

- $y_{jlt}=1$ if task $J_{j,l}$ completes at time t, 0 otherwise

The scheduling problem depicted with the formulation Y-PR is proved to be pseudopolynomially solvable. The linear relaxation lower bound of X-IP is compared (on instances with up to 50 jobs) with the lower bound provided by Y-PR, and X-IP appears to provide stronger lower bounds.

Sahin and Ahuja [27] introduce Y-PR in order to compute lower bounds, as the classical preemption does not provide lower bounds for $1|r_i|\sum v(J_i)$. For $1|r_i|\sum V_k$, Y-PR has no interest, as a better lower bound is computed in polynomial time with SRPT algorithm (the relaxed problem $1|r_i, pmtn|\sum V_k$ where preemption is allowed is discussed in Chapter 2).

From Y-PR, the authors derive another formulation, Y-NP, in order to compute an optimal solution. In Y-NP, all the unit-time tasks of the same parent job must be processed subsequently one after each other, which can be expressed with constraint 1.9.

$$y_{jlt}=y_{j,l+1,t+1}, \qquad l\in\{1,\dots,p_j-1\}, j\in\{1,\dots,N\}, t\in\{1,\dots,T\} \tag{1.9}$$

Moreover, only the last unit-time task of each job has a cost, equal to the cost of the parent job. Then, the Y-NP formulation is:

$$(Y-NP)\begin{cases} \min \sum_{j=1}^{N}\sum_{t=1}^{T} c_{jt}y_{jp_jt} & (1.10)\\ \text{s.t. Constraints (1.6) – (1.9)} \end{cases}$$

X-IP and Y-NP are compared, on instances with up to 50 jobs, and subject to a time limit. It appears that X-IP finds more often the optimal solution than Y-NP. This is probably due to the fact that Y-PR is less compact than X-IP, as the number of variables in Y-PR is $\sum_{j=1,\dots,N} p_j$ times the number of variables in X-IP.

Sahin and Ahuja [27] also provide some heuristic methods. Two methods are based on the result of the linear relaxation of X-IP. The first method, called LPA-α and based on

the notion of α-points, is described as follows. Let $\overline{x}_{jt}$ represent the values of the decision variables in the solution, $j \in \{1, \ldots, N\}$, $t \in \{1, \ldots, T\}$. For each job J_j, let $\bar{t}_j$ be the completion time corresponding to its α-point: $\bar{t}_j = \arg\min_{t \in \{1,\ldots,T\}} \left\{ \sum_{t'=1}^{t} \overline{x}_{jt'} = \alpha \right\}$. A feasible solution is obtained by ordering the jobs by their $\bar{t}_j$. LPA-α is tested with $\alpha \in \{\epsilon, 0.25, 0.5, 0.75, 1\}$, where $\epsilon > 0$ and sufficiently small.

The second method, LPT, is similar to LPA-α, the only difference is in the way of computing $\bar{t}_j$: $\bar{t}_j = \sum_{t'=1}^{T} t\overline{x}_{jt'}$.

Finally, they present a k-exchange neighborhood (k-opt) algorithm. This algorithm starts from an initial feasible solution, with a fixed integer k. For each position i from 1 to $N - k$, the jobs at positions $i, i+1, \ldots, i+k$ are considered. $k!$ neighboring solutions are constructed, by replacing the sequence $i, i+1, \ldots, i+k$ by all the possible sequences of the jobs at those positions. The neighbor solution with the minimal cost becomes the current solution. This method is tested in the following way: first, perform 3-exchanges passes until no further improvement is possible. Then, on the obtained solution, perform 4-exchanges passes until no further improvement is possible.

All these algorithms are fast, but for the first two methods we have to solve the linear relaxation of X-IP. The algorithm that gives the best results is LPT-α, with gaps up to 3%, but k-neighborhood is the fastest (since it does not need the preliminary solving of X-IP), and also gives good solutions, with gaps up to 6%.

Detienne et al. [7] consider the unrelated parallel machines problem with release dates $R|r_i|\sum_{i=1}^{N} v(J_i)$. They present an industrial application of the problem in semiconductor manufacturing processes, and provide an ILP formulation for the special case without release dates. In the ILP model, they introduce the notion of occurrences of jobs: a job has as many occurrences as there are moments of value changes (called jump points) in its stepwise cost function. Hence, each occurrence can be considered as a job with a unique due date. Then, they can exploit the dominance rule that says that if there exists, on a single machine, a schedule where all the jobs meet their due dates, then the schedule where the jobs are ordered in EDD-order is such that all the jobs meet their due dates. Hence, by allowing exactly one occurrence per job to be processed, and by assigning each processed occurrence to exactly one machine, the above dominance rule allows to verify if each processed occurrence meets its due date. The experiments show that the formulation solves in a reasonable time instances with up to 50 jobs and 9 jump points per job.

For the general case with release dates, which is more closely related to $1|r_i|\sum V_k$, the EDD order on occurrences is no more dominant, because of the release dates. Hence, Detienne et al. [7] define another (strict partial) order, $\prec$, on the occurrences, defined as follows: given two occurrences i_l (associated to job i) and j_q (associated to job j), $i_l \prec j_q \Leftrightarrow (d_{i_l} < d_{j_q})$ or $(d_{i_l} = d_{j_q}$ and $r_i < r_j)$, where d_{i_l} (resp. d_{j_q}) is the jump point associated to occurrence i_l (resp. j_q). This order becomes dominant by introducing

some virtual jump points. For each occurrence i_l associated to job i and jump point d_{i_l}, and for each occurrence j_q of job j (and jump point d_{j_q}) such that $d_{i_l} > d_{j_q}$ and $r_i < r_j$, a new occurrence i_n of job i is created, associated to the virtual jump point $d_{i_n} = d_{j_q}$. Moreover, Detienne et al. [7] describe some rules which allow to detect if a virtual jump point is useless and thus avoid its addition. If we use the proposed model for $1|r_i|\sum_{k=1}^{K} V_k$, every virtual jump point is useless, since:

- if there are two occurrences i_l and j_q such that $r_i < r_j$ and $d_{i_l} > d_{j_q}$, the created virtual jump point for i_l is equal to d_{j_q};
- all the jobs have the same jump points, hence there already exists an occurrence i_q, related to job i, whose due date is d_{j_q}.

The dominant order on the occurrences defined by Detienne et al. [7] is valid for $1|r_i|\sum_{k=1}^{K} V_k$ and is equivalent to the dominance rule that we introduce in Chapter 3 (ERD-schedule dominance, p. 34).

Detienne et al. [7] consider the minimization version of the stepwise objective function defined at the beginning of Section 1.3.1 (p. 13), i.e. where $w_{k,i}$ represent costs. They assume that $w_{1,i} = 0$, $i = 1, \ldots, N$. The model of Detienne et al. [7], called $QTS^{\prec}$, follows.

$$
(QTS^{\prec}) \begin{cases}
\min \displaystyle\sum_{\mu \in M} \sum_{q \in \{1,\ldots,\lambda\}} w_q u_q^\mu & & (1.11)\\
\text{s.t.} \displaystyle\sum_{\mu \in M} \sum_{q \in G_i} u_q^\mu = 1, & i \in J & (1.12)\\
t_q^\mu \geq t_{q-1}^\mu + p_{\sigma(q)}^\mu u_q^\mu, & \mu \in M, q \in \{1,\ldots,\lambda\} & (1.13)\\
t_q^\mu \geq (r_{\sigma(q)} + p_{\sigma(q)^\mu}) u_q^\mu, & \mu \in M, q \in \{1,\ldots,\lambda\} & (1.14)\\
t_q^\mu \leq d_q, & \mu \in M, q \in \{1,\ldots,\lambda\} & (1.15)\\
u_q^\mu \in \{0,1\}, & \mu \in M, q \in \{1,\ldots,\lambda\} & (1.16)\\
t_q^\mu \geq 0, & \mu \in M, q \in \{1,\ldots,\lambda\} & (1.17)
\end{cases}
$$

where:

- M is the set of machines
- λ is the total number of occurrences, numbered w.r.t. the dominant order described above.
- d_q is the jump point associated with occurrence q (its "due date")
- w_q is the cost associated with occurrence q
- $\sigma(q)$ denotes the job corresponding to the q-th occurrence
- $p_{\sigma(q)}^\mu$ is the processing time of job $\sigma(q)$ on machine μ

- $r_{\sigma(q)}$ is the release date of job $\sigma(q)$
- $G_i = \{q \in \{1, \ldots, \lambda\} | \sigma(q) = i\}$ is the set of occurrences linked with job J_i
- u_q^μ is equal to 1 if occurrence q is selected for machine μ, and to 0 otherwise
- t_q^μ denotes the completion time of occurrence q on machine μ

As Detienne et al. [7] remark, the constants of constraints 1.13 and 1.14 degrade the quality of the linear relaxation of the model, and yet this model cannot solve large instances in reasonable time. In the Branch and Bound method that we present in Chapter 3, we use a similar branching structure as in $QTS^\prec$ (since it relies on the same dominant order), but we consider constructive bounds that allow fast results on big instances.

Starting from $QTS^\prec$, we deduce a model for $1|r_i|\sum_{k=1}^{K} V_k$: $QTS^\prec(\sum V_k)$, where each job J_i has K occurrences $o_i^1, \ldots, o_i^K$. Each occurrence o_i^k is associated with delivery date D_k, $k = 1, \ldots, K$. The dominant order on these occurrences is the same as for Detienne et al. [7]: $o_i^k \prec o_j^{k'} \Leftrightarrow k < k'$ or $(k = k'$ and $i < j)$.

$$
(QTS^\prec(\textstyle\sum V_k)) \begin{cases}
\max \displaystyle\sum_{i\in\{1,\ldots,N\}} \sum_{k\in\{1,\ldots,K\}} (K-k+1)u_i^k & (1.18)\\
\text{s.t.} \displaystyle\sum_{k=1}^{K} u_i^k = 1, \qquad i \in \{1,\ldots,N\} & (1.19)\\
t_1^k \geq t_N^{k-1} + p_1 u_1^k, \qquad k \in \{2,\ldots,K\} & (1.20)\\
t_i^k \geq t_{i-1}^k + p_i u_i^k, \qquad i \in \{2,\ldots,N\}, k \in \{1,\ldots,K\} & (1.21)\\
t_i^k \geq (r_i + p_i) u_i^k, \qquad i \in \{1,\ldots,N\}, k \in \{1,\ldots,K\} & (1.22)\\
t_i^k \leq D_k, \qquad i \in \{1,\ldots,N\}, k \in \{1,\ldots,K\} & (1.23)\\
u_i^k \in \{0,1\}, \qquad i \in \{1,\ldots,N\}, k \in \{1,\ldots,K\} & (1.24)\\
t_i^k \geq 0, \qquad i \in \{1,\ldots,N\}, k \in \{1,\ldots,K\} & (1.25)
\end{cases}
$$

In order to provide a more efficient model, Detienne et al. [7] consider the following property.

Property. Given a sequence of occurrences $\delta_1, \ldots, \delta_n$ on a given machine, all the occurrences meet their due dates if and only if: $r_{\sigma(\delta_q)} + \sum_{j=q}^{l} p_{\sigma(\delta_j)} \leq d_{\delta_l}$, $q \in \{1,\ldots,n\}$, $l \in \{q,\ldots,n\}$.

Hence, in the $QTS^\prec$ model, Constraints (1.13), (1.14), (1.15) and (1.17) are replaced by constraint (1.26) below.

$$
(r_{\sigma(q)} + p_{\sigma(q)}^\mu) u_k^\mu + \sum_{j=q+1}^{l} p_{\sigma(l)}^\mu u_l^\mu \leq d_l, \qquad \mu \in M, q \in \{1,\ldots,\lambda\}, l \in \{q,\ldots,\lambda\} \qquad (1.26)
$$

The obtained model, $QTS^{\prec}_{MMKP}$, can be seen, as remarked by Detienne et al. [7], as a Multiple-choice Multidimensional Knapsack Problem (MMKP) [17], where:

- each group of items represents one job,
- each item corresponds to one occurrence of a job,
- each resource corresponds to the available processing time between one release date and one jump point.

Since this model has a very large number of constraints, Detienne et al. [7] solve it with a constraint generation scheme, which solves, within the time limit of 1000 s, more than 80% of the instances with up to 60 jobs, and more than 60% of the instances with 70 to 100 jobs.

Although the results of Detienne et al. [7] and of Sahin and Ahuja [27] are valid for $1|r_i|\sum_{k=1}^{K} V_k$, our contribution is to analyze the complexity of some problems dealing with common stepwise functions and to propose dedicated exact and heuristic methods based on some structural properties of the optimal solutions.

1.3.2 Other related objective functions

Finally, two other kinds of criteria are related to $\sum V_k$, regarding the presence of a set of common delivery or due dates. First, Hall et al. [12] study the class of problems with fixed delivery dates. They consider several classical scheduling criteria, always including the following variant: the cost of a job J_i depends on the earliest delivery date occurring after the completion of J_i. In addition, complexity results are established for several problems, with different criteria and machine configurations. Second, there exists another class of problems related to the common due dates: the generalized due date problem [13], where due dates are not related to the jobs. Instead, global due dates are defined, and before each of them, one job must complete. Then, given a schedule, the i-th scheduled job is related to the i-th due date, and its cost is computed in relation to that due date, as for classical due dates. Complexity results have been established by Hall et al. [13] for this class of problems. These two classes of problems differ from $1|r_i|\sum_{k=1}^{K} V_k$, as their payoffs are not cumulative with respect to the delivery dates.

1.4 Conclusion

In this chapter, we illustrated a real world digitization workflow issue. We modeled this problem as a scheduling problem with a new criterion, involving common delivery dates for the jobs. Each job has its own processing time and release date, and the objective is to attain some target quantities (fixed by the client) of digitized books at each delivery date, while maximizing the payoff of the manufacturer. This induces a cumulative aspect,

since each job can be counted several times (once for each delivery date subsequent to its completion time). The same objective function can alternatively be represented as a sum of stepwise payoff functions associated to the jobs. We also presented a few works dealing with job stepwise cost functions in the scheduling literature.

In the next chapter, we establish the complexity of the single machine problem, in the general case and in some particular cases.

Chapter 2

Complexity analysis for the single machine problem

In this chapter, we establish the complexity of $1|r_i|\sum_{k=1}^{K} V_k$, for an arbitrary K and for $K = 2$. We first prove the strong NP-hardness of $1|r_i|\sum_{k=1}^{K} V_k$ (Section 2.1), and the NP-hardness of the two delivery dates problem $1|r_i|V_1 + V_2$ (Section 2.2). Finally, in Section 2.3 we present some polynomial cases, including the Single Delivery Date problem $1|r_i|V$ (Section 2.3.2), whose resolution method is widely used in the next chapters.

NOTATIONS. Given a set of jobs $\mathcal{J}$, we denote by $p(\mathcal{J})$ its total processing time: $p(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} p_i$. Given a sequence Γ of jobs (resp. a schedule S), $\mathcal{J}(\Gamma)$ (resp. $\mathcal{J}(S)$) denotes the set of the jobs of Γ (resp. S). Given a sequence Γ of jobs (resp. a (partial) schedule S), we denote by $p(\Gamma)$ (resp. $p(S)$) the total processing time $p(\mathcal{J}(\Gamma))$ (resp. $p(\mathcal{J}(S))$).

2.1 The multiple delivery dates problem

In order to prove the NP-hardness of problem $1|r_i|\sum_{k=1}^{K} V_k$, we prove the NP-completeness of the corresponding decision problem, defined as follows.

MDD (Multiple Delivery Dates problem). Given a set of N jobs $J_1, \ldots, J_N$, K delivery dates $D_1, \ldots, D_K$, and a value $\mathcal{V}$, does there exist a schedule S such that $v(S) \geq \mathcal{V}$?

To prove the NP-completeness of MDD, we show that the 3-Partition problem, defined as follows [10], reduces to MDD:

3-PARTITION. Given positive integers m, B, and a set of integers $A = \{a_1, a_2, \ldots, a_{3m}\}$ such that $\sum_{i=1}^{3m} a_i = mB$ and $B/4 < a_i < B/2$ for $1 \leq i \leq 3m$, does there exist a partition $\langle A_1, A_2, \ldots, A_m \rangle$ of A into 3-element sets such that, for each i, $\sum_{a \in A_i} a = B$?

Before proving the NP-completeness of MDD, we need to introduce the following definition.

Definition 1. *In a schedule S, a straddling job J_i is a job such that $C_i - p_i < D_k < C_i$, for some $k \in \{1, \ldots, K\}$.*

Theorem 1. *MDD is unary NP-complete.*

Proof. Given a feasible solution S for MDD, its payoff $v(S)$ can be computed in $O(N)$ time, thus $MDD \in NP$.

We show that 3-Partition reduces to MDD. Suppose we are given an instance of 3-Partition, with m, B and A. The corresponding input to MDD is a set of $N = 4m$ jobs, a set of m delivery dates, and a value $\mathcal{V} = 2m(m+1)$.

The delivery dates are: $D_j = j(3mB + B + 1)$, $j \in \{0, \ldots, m\}$.

We define two different kinds of jobs, with processing times and release dates specified as follows. For each $i \in \{1, \ldots, 3m\}$, job $\tilde{J}_i$ has a processing time of $mB + a_i$, and its release date is zero; for each $j \in \{1, \ldots, m\}$, job $\overline{J}_j$ has a processing time of 1, and its release date is $D_j - 1$.

We show that MDD has a solution with a payoff at least equal to $\mathcal{V}$ if and only if the desired partition of A exists.

First, if there exists a partition $\langle A_1, A_2, \ldots, A_m \rangle$ of A, such that for each i, $\sum_{a \in A_i} a = B$, then the following schedule σ is such that $v(\sigma) = \mathcal{V}$ (see Figure 2.1).

For each $j \in \{1, \ldots, m\}$:

- let $A_j = \{a_{1_j}, a_{2_j}, a_{3_j}\}$; then the three jobs $\tilde{J}_{1_j}, \tilde{J}_{2_j}, \tilde{J}_{3_j}$, of length $mB + a_{1_j}$, $mB + a_{2_j}$, $mB + a_{3_j}$ respectively, are scheduled from D_{j-1} to $D_{j-1} + (3m+1)B = D_j - 1$.
- the job $\overline{J}_j$ is scheduled from $D_j - 1$ to D_j.

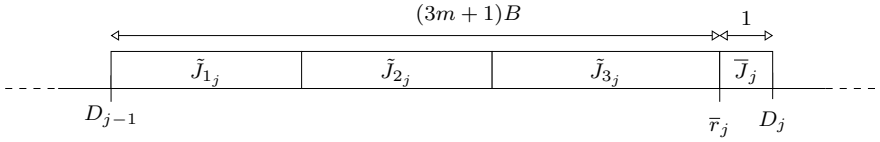


Figure 2.1: Schedule σ between D_{j-1} and D_j, $j = 1, \ldots, m$.

In schedule σ, 4 jobs are executed between each pair of consecutive delivery dates. Hence, $v(\sigma) = 4(1 + 2 + \cdots + m) = 4m(m+1)/2 = \mathcal{V}$.

Conversely, assume that there exists a schedule σ' such that $v(\sigma') \geq \mathcal{V}$.

Let us show that an upper bound on the payoff earned by the $\tilde{J}$-type jobs in σ' is $3m(m+1)/2$. For this purpose, let us consider all the partial schedules in which the

$\tilde{J}$-type jobs are scheduled as soon as possible, i.e. starting from 0 and completing at $D_m - m$, without idle times (we do not consider $\overline{J}$-type jobs for now), as in Figure 2.2.

Notice that the processing time of each $\tilde{J}$-type job is strictly greater than $mB + B/4$. Then, in each of the partial schedules defined above, we have: for every $j \in \{0, \ldots, m-1\}$, the $(3j+1)$-th job of the schedule completes after $(3j+1)(mB+B/4) = 3jmB+3jB/4+mB+B/4 > 3jmB+jB+j = D_j$. Hence, the $(3j+2)$-th and $(3j+3)$-th jobs also complete after D_j. Therefore, the payoff of each of these three jobs is at most $(m-j)$. Hence, the total payoff is at most $\sum_{j=0}^{m-1} 3(m-j) = 3m(m+1)/2$.

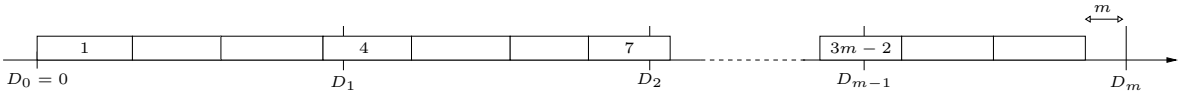


Figure 2.2: Computing an upper bound on the payoff earned by the $\tilde{J}$-type jobs. The $(3j+1)$-th jobs are designed by their rank in the schedule.

However, since $v(\sigma') \geq 2m(m+1)$, a payoff of at least $m(m+1)/2$ is earned by the $\overline{J}$-type jobs in σ'. The only way for these jobs to earn this payoff, is to schedule one of them in each interval $[D_j - 1, D_j]$, which yields a payoff of exactly $m(m+1)/2$. Hence, for every $j \in \{1, \ldots, m\}$, the job $\overline{J}_j$ is scheduled from $D_j - 1$ to D_j in σ'

We deduce that the $\tilde{J}$-type jobs are scheduled in the intervals $[D_{j-1}, D_j - 1]$ in σ'. More precisely, exactly three $\tilde{J}$-type jobs are scheduled in each interval $I'_j = [D_{j-1}, D_j - 1]$, $j = 1, \ldots, m$. Indeed, their processing time is strictly greater than mB, therefore at most three of them can be scheduled in each interval I'_j (of length $(3m+1)B$). Hence, exactly three jobs are scheduled in each interval I'_j, otherwise $v(\sigma') < \mathcal{V}$.

Each $\tilde{J}$-type job can be seen as composed of two parts: the first of length mB, and the second of length a_i. Since three $\tilde{J}$-type jobs are scheduled in I'_1, their first parts occupy $3mB$ time slots. The remaining slots of I'_1, of total length B, are occupied by the second parts of $\tilde{J}$-type jobs, each one being of length a_i. The same reasoning holds for every I'_j, $j = 1, \ldots, m$. Therefore, the second parts of $\tilde{J}$-type jobs, each one of length a_i, are partitioned into m groups, each one having a total length of B, which constitutes a solution to 3-Partition. □

2.2 The two delivery dates problem

We consider the problem of $1|r_i|\sum_{k=1}^{K} V_k$ with $K = 2$, i.e. $1|r_i|V_1+V_2$. Its corresponding decision problem is the following:

2DD (Two Delivery Dates problem). Given a collection of N jobs: $\{J_1, \ldots, J_N\}$, two delivery dates: D_1, D_2, and a value $\mathcal{V}$, does there exist a schedule S such that $v(S) \geq \mathcal{V}$?

We show a polynomial reduction from the Partition problem to 2DD. A definition of the Partition problem follows.

Partition. Given n positive integers $s_1, \ldots, s_n$, does there exist $L' \subset L = \{1, \ldots, n\}$ such that: $\sum_{i \in L'} s_i = \sum_{i \in L \setminus L'} s_i$?

Theorem 2. *2DD is NP-complete.*

Proof. Given a feasible solution S for 2DD, its payoff $v(S)$ can be computed in $O(N)$ time, thus $2DD \in NP$.

We show that Partition reduces to 2DD. Suppose we are given an instance of Partition, with n positive integers $s_1, \ldots, s_n$. Let $b = \frac{1}{2} \sum_{i=1}^{n} s_i$. The corresponding input of 2DD is a set of $3n$ jobs, two delivery dates and a value $\mathcal{V} = 5n$.

There are three different kinds of jobs, with processing times and release dates specified as follows. For each $i \in \{1, \ldots, n\}$:

- job $\tilde{J}_i$ has a processing time of $nb + s_i$, and its release date is zero,
- job $\widehat{J}_i$ has a processing time of nb, and its release date is zero,
- job $\overline{J}_i$ has a processing time of 1, and its release date is $b(n^2 + 1)$.

The delivery dates are: $D_1 = b(n^2 + 1) + n$ and $D_2 = D_1 + b(n^2 + 1)$.

We now show that 2DD has a solution of value at least $\mathcal{V}$ if and only if the desired partition of $s_1, \ldots, s_n$ exists.

First, if Partition has a solution, the solution is represented by a partition of L in two sets L' and $L \setminus L'$. Let q be the cardinality of L'. The cardinality of $L \setminus L'$ is then $n - q$.

Let σ be the following schedule (see Figure 2.3):

1. The q jobs of the set $\{\tilde{J}_i | i \in L'\}$ are executed from time zero to $b(qn + 1)$
2. The $n - q$ jobs of the set $\{\widehat{J}_i | i \in L \setminus L'\}$ are executed from $b(qn + 1)$ to $b(n^2 + 1)$
3. The n jobs $\overline{J}_1, \ldots, \overline{J}_n$ are executed from $b(n^2 + 1)$ to D_1
4. The $n - q$ jobs $\{\tilde{J}_i | i \in L \setminus L'\}$ are executed from D_1 to $D_1 + (n - q)nb + b$
5. The q jobs of the set $\{\widehat{J}_i | i \in L'\}$ are executed from $D_1 + (n - q)nb + b$ to D_2

Therefore, in schedule σ, $q + (n - q) + n = 2n$ jobs are executed before D_1, and $(n - q) + q = n$ jobs complete into $I_2 =]D_1, D_2]$. Hence, $v(\sigma) = 5n$.

Conversely, suppose there exists a schedule σ' whose payoff is at least $5n$. We first prove that, in σ', $2n$ jobs are executed before D_1. Indeed, if less than $2n$ jobs are

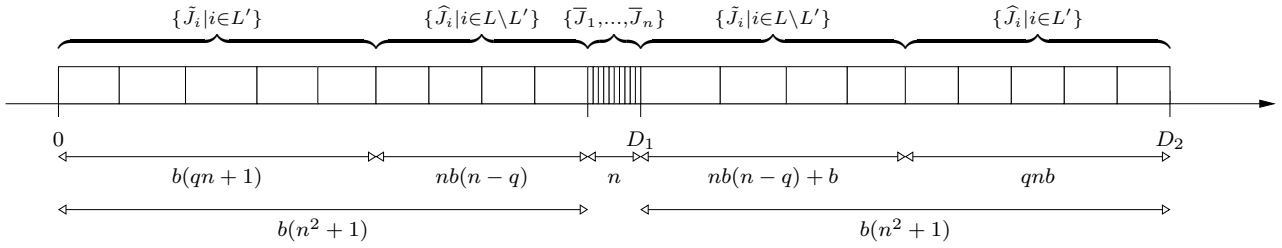


Figure 2.3: Schedule σ.

executed before D_1, then $v(\sigma')$ is less than $5n$ (because of the total number of jobs). At most n $\tilde{J}$-type and $\widehat{J}$-type jobs can be processed before D_1, since the processing time of each of these jobs is at least equal to nb. Hence, the n jobs $\overline{J}_1, \ldots, \overline{J}_n$ must all be executed before D_1 (in the interval $[b(n^2+1), D_1]$), otherwise it would be impossible to execute $2n$ jobs before D_1. Therefore, n $\tilde{J}$-type and $\widehat{J}$-type jobs are executed in the interval $[0, b(n^2+1)]$.

So, in schedule σ', to reach the minimal payoff of $5n$, exactly n jobs are processed in the interval $I_1' = [0, b(n^2+1)]$ and n jobs in the interval $I_2 =]D_1, D_2]$. The length of each of the intervals I_1' and I_2 is $b(n^2+1)$. Moreover, the $2n$ jobs scheduled into these intervals are all the $\tilde{J}$-type and $\widehat{J}$-type jobs, whose processing times are at least equal to nb. Hence, each of them can be seen as composed of two parts: a first part of length nb, and a second part whose length is 0 for $\widehat{J}$-type jobs and s_i for $\tilde{J}$-type jobs. Since n jobs are executed in I_1', $n^2 b$ time slots are occupied by the first parts of these n jobs. The remaining portion of I_1', which is of length b, is occupied by the second parts of $\tilde{J}$-type jobs, each of length s_i. The same reasoning holds for I_2. Therefore, the second parts of the $\tilde{J}$-type jobs, each of length s_i, are partitioned in two groups, each of total length equal to b, which constitutes a solution to Partition. □

In Chapter 4 we present a pseudopolynomial time algorithm for $1|r_i|V_1 + V_2$, proving thus that $1|r_i|V_1 + V_2$ is weakly NP-hard.

2.3 Polynomial cases

We first introduce three simple polynomial cases, and then the Single Delivery Date problem $1|r_i|V$.

2.3.1 Relaxations of the general problem

Three polynomial cases resulting from different relaxations of the general problem $1|r_i|\sum_{k=1}^{K} V_k$ have been identified.

1. The problem with no release date $1||\sum_{k=1}^{K} V_k$ can be easily shown to be optimally solved by sequencing the jobs in a nondecreasing order of their processing times (SPT rule) [15].

2. The preemptive case $1|r_i, pmtn|\sum_{k=1}^{K} V_k$ can be optimally solved using Shortest Remaining Processing Time rule (SRPT), which is also called Modified Smith's rule [2]: at each release time or completion time of a job, the available job with the smallest remaining processing time is scheduled. This can be proved with the same arguments used to show that applying SRPT [2] yields an optimal solution for the problem $1|r_j, pmtn|\sum C_j$. Therefore, the problem $1|r_i, pmtn|\sum_{k=1}^{K} V_k$ can be solved in $O(NlogN)$ time.

3. We consider the case where all the jobs have the same processing time $1|r_i, p_i = p|\sum_{k=1}^{K} V_k$.

   Definition 2. *A "left-shifted" schedule is such that each job starts as soon as possible, either at its release date or at the completion time of the preceding job in the schedule.*[1]

   Hence, to represent a left-shifted schedule S it is sufficient to provide the ordered sequence of all the jobs in S.

   Proposition 1. *A left-shifted schedule in which jobs are ordered w.r.t. a nondecreasing order of their release dates is optimal for the identical processing times problem* $1|r_i, p_i = p|\sum_{k=1}^{K} V_k$.

   Proof. In the following proof, we will only consider left-shifted schedules. Indeed, every feasible schedule can be transformed into a left-shifted schedule in $O(N)$ time, without decreasing its payoff. Let S_R be a schedule in which jobs are ordered w.r.t. a nondecreasing order of their release dates. We show that every feasible schedule S can be transformed into S_R, and that $v(S_R) \geq v(S)$. We renumber the jobs w.r.t. their order in S: $J_1, \cdots, J_N$. If there exists a job J_i such that $C_i(S_R) > C_{i+1}(S_R)$, then by swapping J_i and J_{i+1} in S, the total payoff will not decrease. Indeed, if $r_i = r_{i+1}$, the payoff does not change, since $p_i = p_{i+1}$. Otherwise, if $r_i > r_{i+1}$, the payoff can remain unchanged, or can possibly increase if $C_i(S) - p_i = r_i$ and $[r_i - 1, r_i]$ is an idle time. Iterating this process, we get, after a polynomial number of steps, the schedule S_R such that $v(S_R) \geq v(S)$. Since this result holds for every feasible schedule S, it also holds for every optimal schedule. Hence the proposition holds. □

   Proposition 1 leads to an optimal algorithm for problem $1|r_i, p_i = p|\sum_{k=1}^{K} V_k$ in $O(NlogN)$ time.

[1] Such a schedule is also called a non-delay schedule.

2.3.2 The single delivery date problem

The problem with a single delivery date $1|r_i|V$ is equivalent to the problem $1|r_i, d_i = d|\sum U_i$. In order to solve $1|r_i|V$, we introduce the Single Delivery Date algorithm (SDD-algorithm) that solves the problem SDD_I, defined below. The SDD-algorithm is widely used to establish the results of the next chapters.

NOTATION. Let D be the unique delivery date of $1|r_i|V$.

Definition 3. *SDD_I: given an interval $[I_{low}, I_{up}]$ and a set of jobs $\mathcal{J}$, a feasible solution is a (partial) schedule S where each job $J_i \in \mathcal{J}(S) \subseteq \mathcal{J}$ must start at or after $\max(r_i, I_{low})$ and complete at or before I_{up}. An optimal solution is a feasible solution with the maximum number of jobs in the interval $[I_{low}, I_{up}]$.*

As SDD_I is defined on a finite horizon, there might not exist feasible schedules as defined in Chapter 1 (p. 7) (i.e. that schedule all the jobs of the instance). Hence, in this section, for the sake of simplicity, we call schedule every feasible (partial) schedule of SDD_I.

Notice that $1|r_i|V$ is equivalent to SDD_I when $I_{low} = 0$, $I_{up} = D$ and $\mathcal{J} = \mathcal{J}^{all}$.

Before presenting the SDD-algorithm, let us introduce two lemmas.

Lemma 1. *There exists an optimal schedule for SDD_I such that the last scheduled job completes at time I_{up}.*

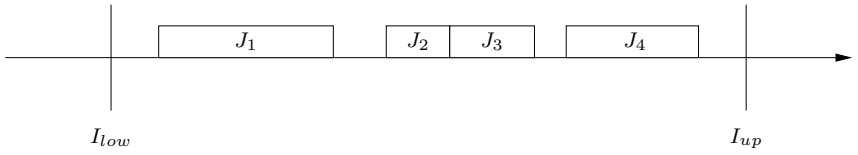
Proof. Let S^* be an optimal schedule for SDD_I such that the last scheduled job J_l does not complete at time I_{up}. We can right-shift J_l such that it completes at time I_{up} (see Figures 2.4a and 2.4b). The obtained schedule schedules as many jobs as S^* into $[I_{low}, I_{up}]$. □

Lemma 2. *There exists an optimal schedule for SDD_I such that there is no idle time between any pair of consecutive scheduled jobs.*

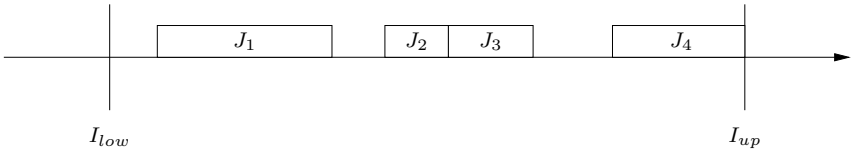
Proof. Let S^* be an optimal schedule such that there are idle times between pairs of consecutive scheduled jobs. Without moving the last job, the other jobs can be right-shifted in order to avoid idle times (see Figures 2.4b and 2.4c). □

Clearly, there exists an optimal schedule for SDD_I such that the last scheduled job completes at time I_{up} and there is no idle time between any pair of consecutive scheduled jobs (see Figure 2.4). Therefore, a feasible schedule for SDD_I can be represented as a sequence of jobs.

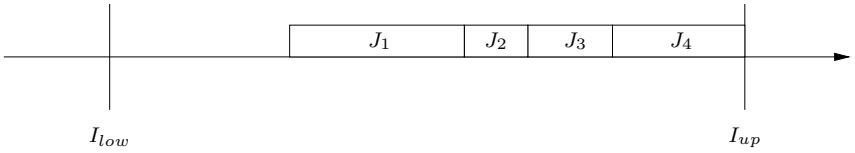
The SDD-algorithm is a polynomial time algorithm solving SDD_I, depicted in Algorithm 1. It is worth noting that it is close to the Moore-Hodgson algorithm [21] for solving $1||\sum U_i$. It is also a particular case of the algorithm of Kise et al. [18] for the problem



(a) A schedule S.



(b) Partial schedule S', obtained from S by right-shifting J_4 in order to make it complete at I_{up}.



(c) Partial schedule S'', obtained from S' by right-shifting J_1, J_2, J_3 in such a way that S'' is a block.

Figure 2.4: Illustration of properties of Lemmas 1 and 2.

$1|r_i|\sum U_i$ where release dates and due dates are agreeable (i.e. where $r_i \leq r_j \Rightarrow d_i \leq d_j$ for each pair of jobs J_i, J_j). Kise et al. [18] provide an $O(N^2)$ algorithm, which can be directly applied to the problem $1|r_i|V$. The algorithm of Kise et al. [18] is based on the same principle as the Moore-Hodgson algorithm, with the difference that the choice of the job to be removed from the current sequence is made in $O(N)$ (by considering all the jobs), while Moore-Hodgson algorithm does it in $O(logN)$ (by extracting the job with the greatest processing time from a heap). Hence, by adapting the algorithm of Kise et al. [18] for $1|r_i|V$ in order to reduce its complexity, we obtain the SDD-algorithm.

NOTATIONS. We will use the following notations, for any sequence Γ of jobs, and any job J_q:

- $\forall J_q \notin \mathcal{J}(\Gamma)$, $J_q.\Gamma$ denotes the sequence obtained by prepending J_q to Γ.
- $\forall J_q \in \mathcal{J}(\Gamma)$, $\Gamma \backslash J_q$ denotes the sequence obtained by removing J_q from Γ, leaving all the other jobs in the order given by Γ.
- $sched(\Gamma)$ denotes the schedule that schedules the jobs of Γ, in the order given by Γ, without idle times and with the last job completing at I_{up}.

In the following, the jobs of $\mathcal{J}$ (input of SDD_I) are reindexed from $i_{|\mathcal{J}|}$ to i_1 w.r.t. ERD order, i.e. $J_{i_{|\mathcal{J}|}} \prec_{ERD} \cdots \prec_{ERD} J_{i_1}$.

At each iteration m ($m = 1, \ldots, |\mathcal{J}|$), the SDD-algorithm constructs a sequence Γ_{i_m} of jobs such that $sched(\Gamma_{i_m})$ is feasible (i.e. each job $J_{i_j} \in \Gamma_{i_m}$ is scheduled into $[\max(r_{i_j}, I_{low}), I_{up}]$). Notice that when $I_{low} = 0$, it is only necessary to check whether release dates are satisfied. Moreover, in each sequence Γ_{i_m}, $m = 1, \ldots, |\mathcal{J}|$, the jobs are ordered w.r.t. ERD order. At the end of the algorithm, $sched(\Gamma_{i_{|\mathcal{J}|}})$ is an optimal schedule.

Input: $\mathcal{J}$, I_{low}, I_{up}
Output: S
1 $\Gamma_{i_0} \leftarrow \emptyset$, $t \leftarrow I_{up}$
2 **for** $m = 1$ **to** $|\mathcal{J}|$ **do**
3 $\Gamma_{i_m} \leftarrow J_{i_m}.\Gamma_{i_{m-1}}$
4 $t \leftarrow t - p_{i_m}$
5 **if** $t < \max\{r_{i_m}, I_{low}\}$ **then**
6 $q \leftarrow \min\{l \mid p_{i_l} = \max\{p_{i_j} | J_{i_j} \in \mathcal{J}(\Gamma_{i_m})\}\}$
7 $\Gamma_{i_m} \leftarrow \Gamma_{i_m} \backslash J_{i_q}$
8 $t \leftarrow t + p_{i_q}$
9 **return** $S \leftarrow sched(\Gamma_{i_{|\mathcal{J}|}})$

Algorithm 1: SDD-algorithm.

An example of the execution of the algorithm on an instance is given in Figure 2.5.

Theorem 3. *The SDD-algorithm solves optimally SDD_I.*

Proof. The arguments for the proof of correctness are similar to those for the Moore-Hodgson algorithm [23]. We give a sketch of proof below.

Let us first show that, at the end of iteration m ($m \in \{1, \ldots, |\mathcal{J}|\}$), $sched(\Gamma_{i_m})$ is feasible (i.e. all its jobs are scheduled into $[I_{low}, I_{up}]$, and satisfy their release dates). Γ_{i_0} is feasible since it is an empty schedule. By induction on m, suppose that Γ_{i_m} is feasible. If $J_{i_{m+1}}.\Gamma_{i_m}$ is feasible, the algorithm constructs $\Gamma_{i_{m+1}}$ as $J_{i_{m+1}}.\Gamma_{i_m}$. Otherwise, the algorithm constructs $\Gamma_{i_{m+1}}$ as $J_{i_{m+1}}.\Gamma_{i_m}$ deprived of its longest job J_{i_q}. Hence, $J_{i_{m+1}}$ starts $p_{i_q} - p_{i_{m+1}}$ time units after the starting time of Γ_{i_m} ($p_{i_q} - p_{i_{m+1}} \geq 0$). Moreover, the starting time of Γ_{i_m} is at least $r_{i_{m+1}}$, since all the jobs of Γ_{i_m} have release dates at least equal to $r_{i_{m+1}}$ (because of the ERD-order). Hence, $J_{i_{m+1}}$ starts not earlier than $r_{i_{m+1}}$. The other jobs of $\Gamma_{i_{m+1}}$ do not start earlier as they started in Γ_{i_m}: the jobs $\{J_{i_j} | j > q\}$ start at the same time, and the jobs $\{J_{i_j} | j < q\}$ start p_{i_q} time units later.

We show next that $\Gamma_{i_{|\mathcal{J}|}}$ is optimal.

Let Γ^m be the sequence $J_{i_m}, \ldots, J_{i_1}$, $m = 1, \ldots, |\mathcal{J}|$. A subsequence E of Γ^m is called eligible if $sched(E)$ is feasible. Consider an eligible subsequence of Γ^m with the maximum number of jobs, and let N_m be its number of jobs. Hence, $N_{|\mathcal{J}|}$ is the value of an optimal solution of SDD_I.

Notice that Γ_{i_m} is an eligible subsequence of Γ^m. In order to prove the optimality of $\Gamma_{i_{|\mathcal{J}|}}$ (i.e. that $\Gamma_{i_{|\mathcal{J}|}}$ has $N_{|\mathcal{J}|}$ jobs), we show by induction on m that Γ_{i_m} has N_m jobs,

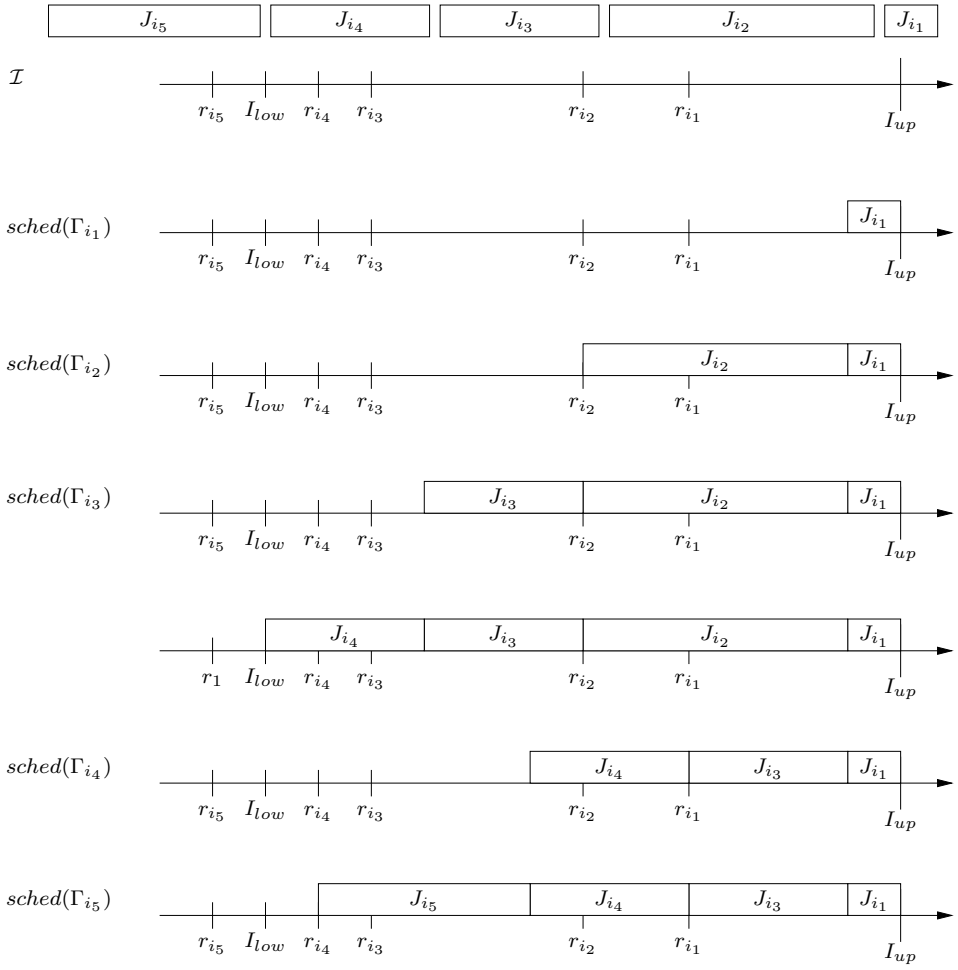


Figure 2.5: Execution of SDD-algorithm on an instance $\mathcal{I}$.

and has the shortest total processing time among the eligible subsequences of Γ^m with N_m jobs.

First, let us show that Γ_{i_1} has N_1 jobs, and that its total processing time is minimal. Γ_{i_1} has N_1 jobs, since if J_{i_1} can start at $I_{up} - p_{i_1}$, the algorithm constructs Γ_{i_1} as the sequence containing only J_{i_1}, otherwise Γ_{i_1} is empty. Clearly, the processing time of Γ_{i_1} is minimal, as Γ^1 only contains job J_{i_1}.

The induction hypothesis is thus that Γ_{i_m} has N_m jobs, and has the shortest total processing time among the eligible subsequences of Γ^m with N_m jobs. We show next that $\Gamma_{i_{m+1}}$ has N_{m+1} jobs, and has the shortest total processing time among the eligible subsequences of Γ^{m+1} with N_{m+1} jobs.

Let us consider the first step of the construction of $\Gamma_{i_{m+1}}$ at iteration $m+1$: job $J_{i_{m+1}}$ is prepended to Γ_{i_m}. There are two cases to be considered.

Case 1. $J_{i_{m+1}}$ is processed into $[\max(r_{i_{m+1}}, I_{low}), I_{up}]$ in $sched(J_{i_{m+1}}.\Gamma_{i_m})$.

We have $N_{m+1} = N_m + 1$, since $|J_{i_{m+1}}.\Gamma_{i_m}| = N_m + 1$ and since $N_{m+1} \leq N_m + 1$ by definition of N_m and N_{m+1}.

Moreover, every eligible subsequence of Γ^{m+1} that contains N_{m+1} jobs, includes job $J_{i_{m+1}}$. Otherwise, it would imply that there exists an eligible subsequence of Γ^m with $N_m + 1$ jobs, which is in contradiction with the inductive hypothesis. Then, to construct $\Gamma_{i_{m+1}}$, we must add job $J_{i_{m+1}}$ to an eligible subsequence of Γ^m with N_m jobs. This eligible subsequence must have a minimal processing time, in order to have $p(\Gamma_{i_{m+1}})$ minimal. All these conditions are satisfied by Γ_{i_m}, by inductive hypothesis. Therefore, $\Gamma_{i_{m+1}} = J_{i_m}.\Gamma_{i_m}$.

Case 2. $J_{i_{m+1}}$ is not processed into $[\max(r_{i_{m+1}}, I_{low}), I_{up}]$ in $sched(J_{i_{m+1}}.\Gamma_{i_m})$.

We have $N_{m+1} = N_m$. Indeed, $J_{i_{m+1}}$ cannot belong to an eligible subsequence E of Γ^{m+1} such that $|E| = N_m + 1$, because by adding $J_{i_{m+1}}$ to Γ_{i_m} (which contains N_m jobs and has the shortest processing time), $J_{i_{m+1}}$ starts earlier than $\max(r_{i_{m+1}}, I_{low})$. Thus, N_m is the maximal number of jobs of an eligible subsequence of Γ^{m+1}.

In order to guarantee the minimality of the total processing time, $\Gamma_{i_{m+1}}$ must contain the N_m shortest jobs of $\mathcal{J}(\Gamma_{i_m}) \cup \{J_{i_{m+1}}\}$. This is done by the algorithm when prepending $J_{i_{m+1}}$ to Γ_{i_m} and then removing the longest job of the obtained sequence. Hence, the algorithm constructs an eligible subsequence $\Gamma_{i_{m+1}}$ with N_{m+1} jobs and minimal processing time. □

The SDD-algorithm runs in $O(NlogN)$ if a heap is used for the search of the job with the largest processing time.

We introduce now some properties of the SDD-algorithm that will be useful in the next chapters.

Property 1. *Given a set $\mathcal{J}$ of jobs and an interval $[I_{low}, I_{up}]$, the SDD-algorithm produces a schedule that schedules the maximal number $N(\mathcal{J}, I_{low}, I_{up})$ of jobs of $\mathcal{J}$ between I_{low} and I_{up}.*

Property 2. *Given a set $\mathcal{J}$ of jobs and an interval $[I_{low}, I_{up}]$, the SDD-algorithm produces a schedule with the shortest processing time $p(\mathcal{J}, I_{low}, I_{up})$ among all the feasible schedules of $N(\mathcal{J}, I_{low}, I_{up})$ jobs of $\mathcal{J}$ between I_{low} and I_{up}.*

The arguments of the proofs of Properties 1 and 2 are similar to those of the proof of Theorem 3.

Property 3. *The SDD-algorithm produces a partial schedule where the jobs are ordered w.r.t. ERD order.*

Property 4. *When SDD-algorithm removes a job from the current sequence, it chooses the job with the longest processing time, and breaks ties by choosing the job with the highest ranking in ERD order (i.e. that follows the other jobs with same longest processing time, in ERD order).*

Properties 3 and 4 are true by construction.

2.4 Conclusion

In this chapter, we showed that the general single machine problem is strongly NP-hard. Moreover, we showed that the single machine problem with two delivery dates is NP-hard (more precisely, the pseudopolynomial algorithm provided in Chapter 4 establishes its weak NP-hardness). Finally, we identified some polynomial cases: among them, the single delivery date case is solved with the SDD algorithm, which is also widely used in the following chapters to establish bounds on the general single machine problem.

In the next chapter, we present a Branch and Bound method based on the structural properties of the problem: dominance rules, dedicated bounds and pruning rules.

Chapter 3

Exact method for the single machine problem: Branch and Bound

In order to solve the problem $1|r_i|\sum_{k=1}^{K} V_k$ in the general case, we present in the current chapter a Branch and Bound method based on the structural properties of the problem. We first introduce the dominance rules (Section 3.1) that are the backbone of the Branch and Bound structure. This structure is based on the branching rule described in Section 3.2. Algorithms for computing lower and upper bounds are described in Section 3.3, while in Section 3.4 some additional pruning rules are presented. Finally, in Section 3.5 we show how to generate random instances and report some numerical results of the Branch and Bound on the generated instances.

Let us first introduce some notations.

NOTATIONS. Any (partial) schedule S of $1|r_i|\sum V_k$ can be split into $K+1$ subschedules $S_1, \ldots, S_{K+1}$; S_k being the subschedule of the jobs completing into $I_k =]D_{k-1}, D_k]$, $k = 1, \ldots, K+1$. Thus, S can be expressed as $S = S_1.S_2. \ldots .S_{K+1}$, where $S_i.S_j$ denotes the concatenation of subschedule S_i with subschedule S_j (assuming that $\mathcal{J}(S_i) \cap \mathcal{J}(S_j) = \emptyset$).

Moreover, for any (partial) schedule S, we denote by $C_{\max}(S_k)$ (resp. $B(S_k)$) the completion time of the last job of S_k (resp. the starting time of the first job of S_k), $k = 1, \ldots, K+1$. If S_k is empty, we set $C_{\max}(S_k) = B(S_k) = D_{k-1}$ (see Figure 3.1).

Finally, for any job J_l and any schedule S, we denote by $b_l(S)$ the starting time of job J_l in schedule S, i.e. $b_l(S) = C_l(S) - p_l$.

3.1 Dominance rules

The following propositions define the dominance rules on which is based the Branch and Bound structure.

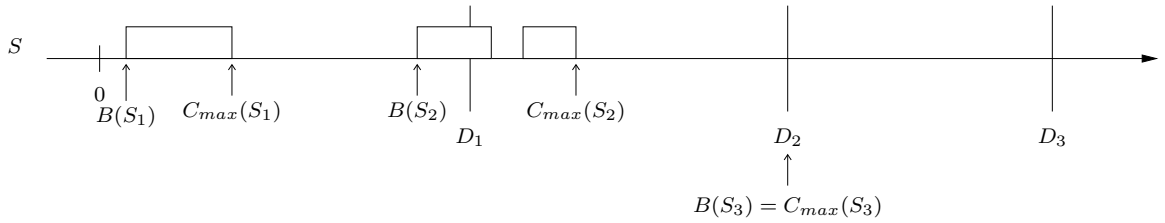


Figure 3.1: $C_{\max}(S_k)$ and $B(S_k)$, $k = 1, \ldots, 3$.

Definition 4. *A feasible schedule $S = S_1.S_2. \ldots .S_{K+1}$ such that all the jobs of S_k ($k = 1, \ldots, K+1$) are scheduled in ERD order is called an ERD-schedule (see Figures 3.2 and 3.3).*

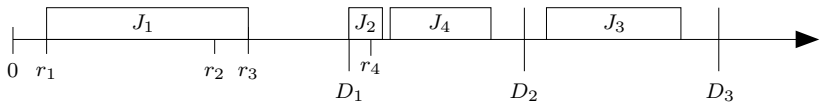


Figure 3.2: An ERD-schedule.

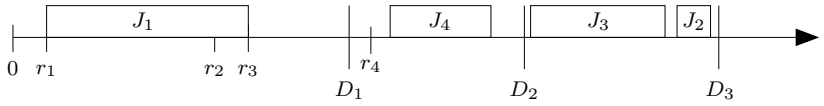


Figure 3.3: This is not an ERD-schedule, since J_2 and J_3 both complete in I_3, but they are not scheduled in ERD order ($r_2 < r_3$).

Proposition 2. *There exists an ERD-schedule that is an optimal solution of $1|r_i|\sum V_k$.*

Proof. We first show that every feasible schedule $S = S_1.S_2. \ldots .S_{K+1}$ can be modified in order to obtain an ERD-schedule S_R such that $v(S_R) \geq v(S)$. Let J_i and J_j be two consecutive jobs in S, such that $D_{k-1} < C_i(S) < C_j(S) \leq D_k$, for some $k \in \{1, \ldots, K+1\}$, and such that $J_j \prec_{ERD} J_i$. A schedule S' can be obtained starting from S, by simply swapping J_i and J_j, such that $C_i(S') = C_j(S)$, $C_j(S') = C_i(S) - p_i + p_j$, and $C_l(S') = C_l(S)$ for every $l \in \{1, \ldots, N\} \backslash \{i, j\}$ (see Figure 3.4).

Let us show that S' is feasible. In S', J_i and J_j are both scheduled in the interval $[b_i(S), C_j(S)]$, whose length is at least $p_i + p_j$. Hence, the starting time of each of the other jobs is the same in S and in S'. Moreover, J_i and J_j meet their release dates in S', since $r_j \leq r_i \leq b_i(S)$ and $r_i \leq b_i(S) < C_j(S) - p_i$.

We show now that $v(S') \geq v(S)$. Let us consider the payoffs of J_i and J_j in both schedules S and S'. $v_{S'}(J_i) = v_S(J_j)$ since $C_i(S') = C_j(S)$. $v_{S'}(J_j) \geq v_S(J_j)$ since

$C_j(S') < C_j(S)$. Therefore, $v_{S'}(J_i) + v_{S'}(J_j) \geq v_S(J_i) + v_S(J_j)$. Notice that $v_{S'}(J_j) > v_S(J_j)$ implies that J_j belongs to S'_q, for some $q \in \{1, \ldots, k-1\}$ (see Figure 3.5).

Iterating this process, we get after a finite number of steps an ERD-schedule S_R. In particular, this modification process can be applied to any optimal schedule. Hence the result holds. □

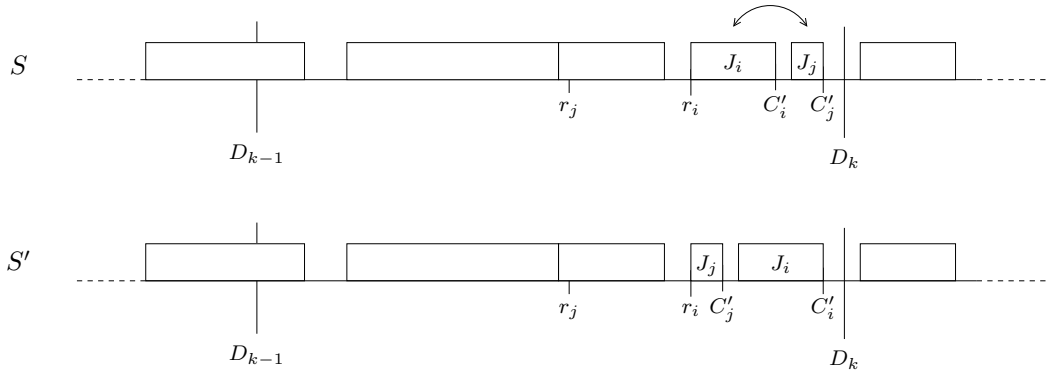


Figure 3.4: An example of swap of J_i and J_j where $v(S') = v(S)$.

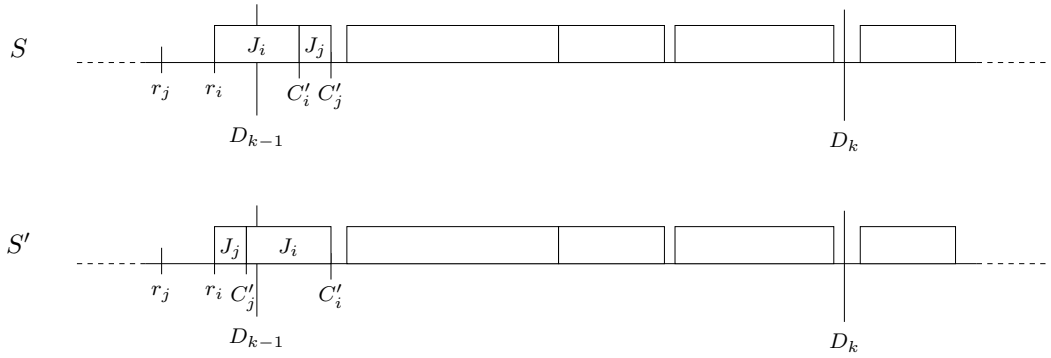


Figure 3.5: An example of swap of J_i and J_j where $v(S') > v(S)$.

Proposition 3. *There exists an optimal schedule for* $1|r_i|\sum V_k$ *where, for* $k \in \{2, \ldots, K\}$, S_k *is a block.*

Proof. Let $S = S_1.S_2. \ldots .S_{K+1}$ be a feasible schedule where there are idle times in S_k, for some $k \in \{2, \ldots, K\}$. Without modifying $C_{max}(S_k)$, we can right-shift the jobs of S_k (except the last one) until there are no more idle times in S_k (see Figure 3.6). We obtain a solution with payoff $v(S)$, since for every $l \in \{1, \ldots, K\} \backslash \{k\}$, the completion times of the jobs of S_l are unchanged, and the jobs of S_k still complete in the interval I_k. In particular, this applies for any optimal schedule. Hence the result holds. □

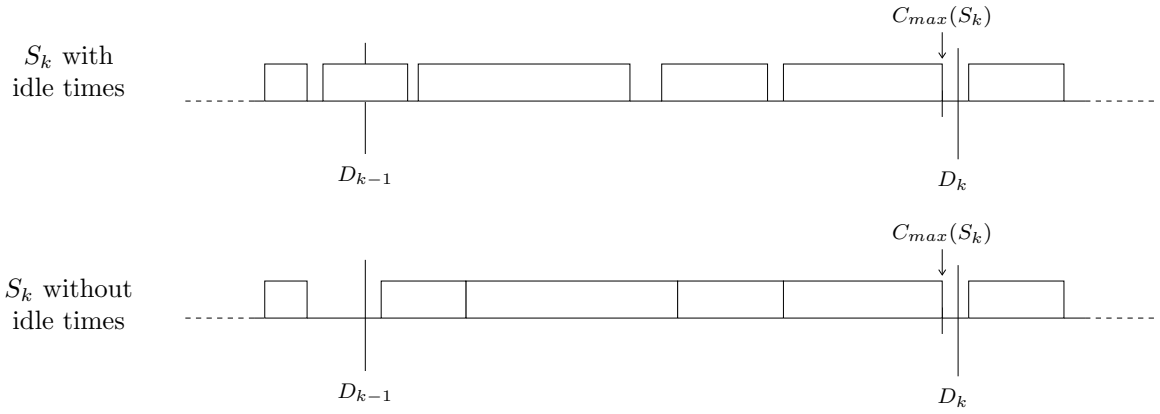


Figure 3.6: Example for Proposition 3.

Obviously, there exists an optimal schedule $S^* = S_1^*.S_2^*. \ldots .S_{K+1}^*$ that is an ERD-schedule where for all $k \in \{2, \ldots, K\}$ S_k^* is a block.

3.2 Branching rule

A Branch and Bound method is an enumerative algorithm that explores the feasible solutions of a problem by constructing them iteratively by the means of a tree [3]. We develop a Branch and Bound algorithm where each node of the tree represents a (partial) schedule. The root node represents the empty schedule, i.e. where no jobs are scheduled; the leaves of the tree represent complete schedules; while all the other nodes represent partial schedules. Each node n^a of the tree, representing a (partial) schedule S^a, is obtained from its father node $n^{f(a)}$ (representing a (partial) schedule $S^{f(a)}$), by adding a job to $S^{f(a)}$, thus obtaining S^a. Since there exists an optimal schedule that is an ERD-schedule, the jobs will be considered in the ERD order. Hence, for the sake of readability, the jobs are, from now on, reindexed in ERD order, thus obtaining after reindexation: $J_1 \prec_{ERD} J_2 \prec_{ERD} \ldots \prec_{ERD} J_N$. We call depth of a node the number of its predecessors (the depth of the root node is 0). Then, the children of a node n^a of depth i are obtained by inserting job J_{i+1} into S^a. J_{i+1} can be inserted into S^a at different positions (described below), hence n^a has a child node for each of these positions. Let us describe precisely these positions. Notice that S^a schedules the jobs $J_1, \ldots, J_i$, which all precede J_{i+1} in ERD order, and that when J_{i+1} is inserted into S^a, it completes into some interval I_k, $k = 1, \ldots, K+1$. Hence, J_{i+1} must be scheduled after the jobs of S^a that complete into I_k, in order to maintain the ERD-schedule structure. Therefore, choosing the position of J_{i+1} in the schedule amounts to simply choosing in which interval I_k $(k = 1, \ldots, K+1)$ J_{i+1} completes. The structure of the Branch and Bound tree is illustrated in Figure 3.7.

When J_{i+1} is added to S^a in order to complete into I_k, J_{i+1} is always scheduled as

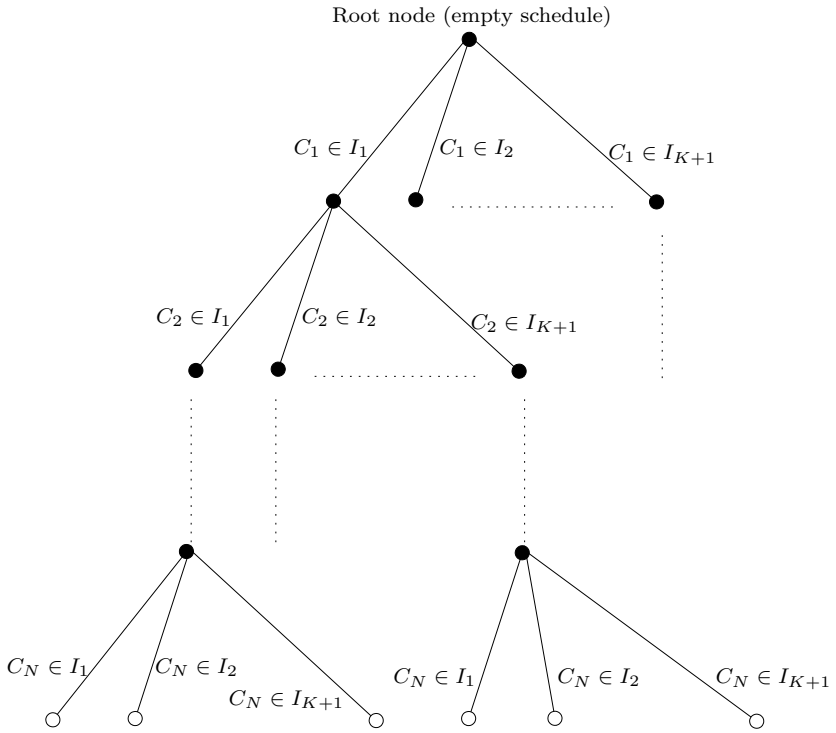


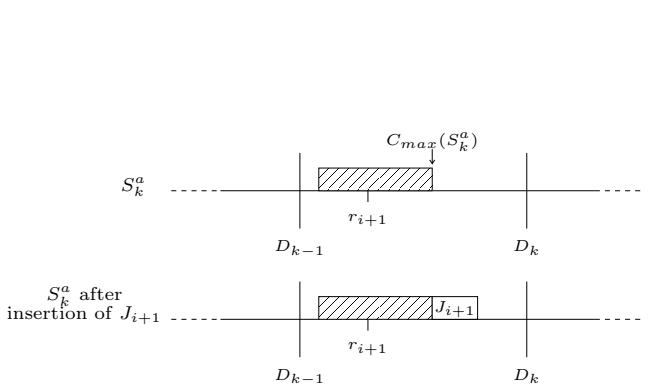
Figure 3.7: Structure of the Branch and Bound tree.

soon as possible after the jobs already completing into I_k (see Figure 3.8). Moreover, if the addition of J_{i+1} induces an idle time between J_{i+1} and the jobs of S^a that complete into I_k, these jobs are right-shifted (as described in the proof of Proposition 3, p. 35), in order for S_k to be a block (see Figure 3.8b). Notice that the insertion of J_{i+1} into S^a_k may require right-shifting the jobs of $S^a_{k'}$, $k' > k$, as shown in Figure 3.8c. However, these right-shiftings are only allowed if they do not decrease the payoff of the right-shifted jobs, i.e. all the jobs of $S^a_{k'}$, $k' > k$, must still complete into $I_{k'}$.

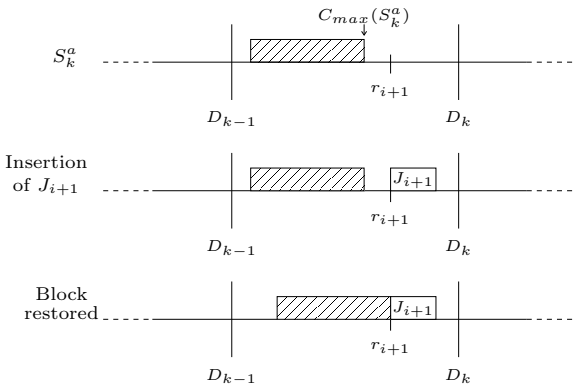
For some intervals I_k, $k = 1, \ldots, K$, J_{i+1} cannot be inserted into S^a_k. In this case, no node is added to the search tree. Figure 3.9 illustrates some examples of these cases. Nonetheless, any job J_{i+1} can always be inserted into S^a_{K+1}. Therefore, each node of the tree has at least one child node and at most $K + 1$ children nodes (one for each interval I_k, $k = 1, \ldots, K + 1$).

The structure of the tree ensures that every ERD-schedule is explored, and among them there is at least an optimal schedule, as proven in Section 3.1. A complete example of a Branch and Bound tree is illustrated in Figure 3.10 for a small instance.

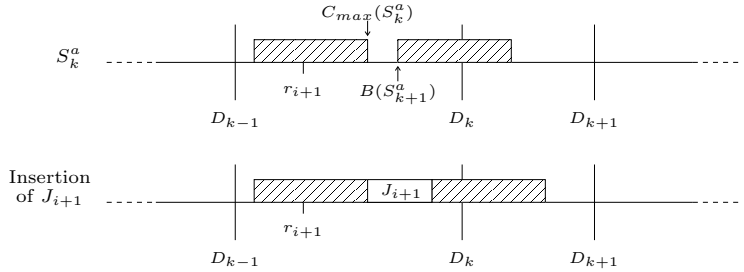
The choice of this branching rule will be justified after presenting the computation of the bounds (p. 53).



(a) $C_{max}(S_k^a) \geq r_{i+1}$.

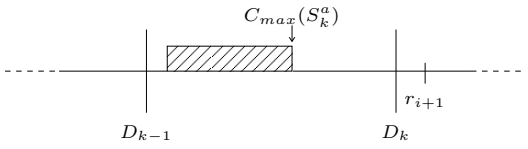


(b) $C_{max}(S_k^a) < r_{i+1}$.

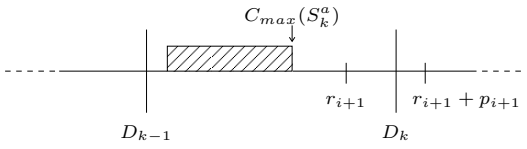


(c) $\max(C_{max}(S_k^a), r_{i+1}) + p_{i+1} > B(S_{k+1}^a)$.

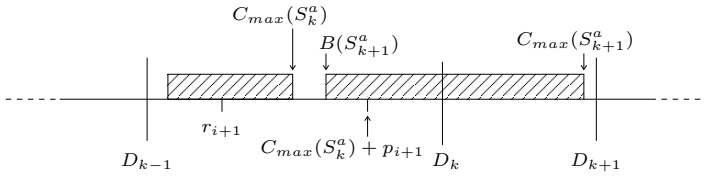
Figure 3.8: Different cases when adding job J_{i+1} to S^a.



(a) $r_{i+1} \geq D_k$.



(b) $r_{i+1} + p_{i+1} > D_k$.



(c) $\max(C_{max}(S_k^a), r_{i+1}) + p_{i+1} - B(S_{k+1}^a) > D_{k+1} - C_{max}(S_{k+1}^a)$.

Figure 3.9: Some cases where job J_{i+1} cannot be added to S^a, while completing into I_k.

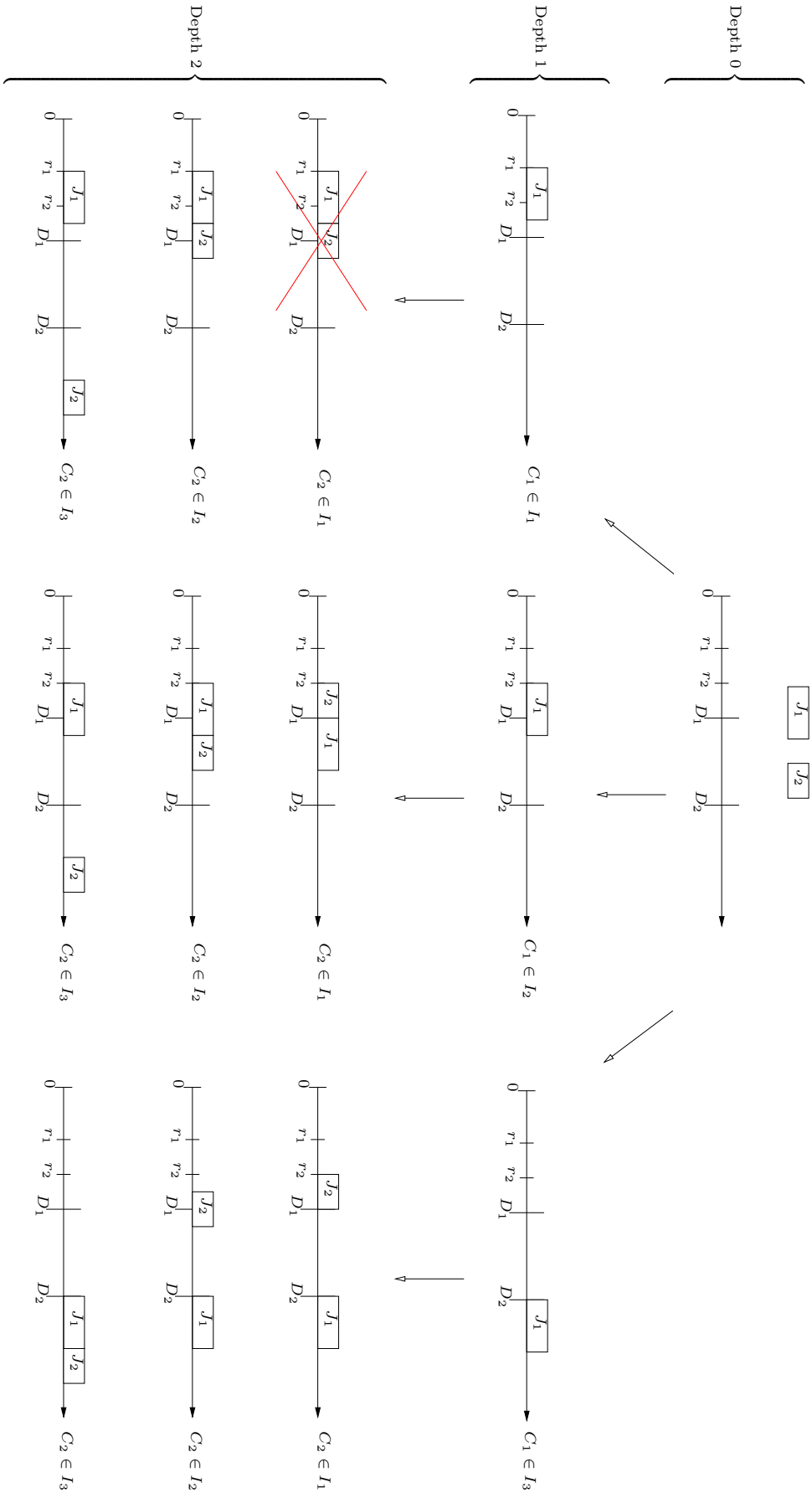


Figure 3.10: Example of Branch and Bound tree.

3.3 Bounds

We have seen in the previous section the structure of the Branch and Bound tree, which enables the exploration of every feasible ERD-schedule. However, in order to find more quickly an optimal schedule, we can prune some branches of the tree when we are sure that they are not useful to find an optimal schedule. For this purpose, each time we create a node n^a in the tree, we compute an upper bound $u\_bound(S^a)$ on the payoffs of the schedules that can be obtained by completing the partial schedule S^a associated to n^a. Equivalently, $u\_bound(S^a)$ is an upper bound on the payoffs of the schedules associated to the descendant nodes of n^a. This way, if a feasible schedule σ with payoff $v(\sigma) \geq u\_bound(S^a)$ is known, node n^a does not need to be developed. Indeed, it will not lead to the construction of a better solution than σ. Therefore, n^a is pruned. In order to perform this kind of pruning, a lower bound on the optimal payoff is updated during the algorithm. At the beginning, an initial lower bound is computed. The payoff of any feasible schedule is a lower bound on the optimal payoff. Afterwards, each time a feasible schedule whose payoff is greater than the current lower bound is constructed, the lower bound is updated to this greater payoff. Therefore, we need to calculate an initial lower bound, and an upper bound at each node of the search tree.

3.3.1 Initial lower bound

As said above, the lower bound allows prunings within the search tree. Therefore, the better the initial lower bound, the faster the exploration of the tree. Let us consider the following idea to quickly construct a feasible schedule with a satisfactory payoff. First, by using SDD-algorithm (p. 29), schedule as many jobs as possible into $[0, D_1]$. The jobs completing in this interval are those providing the greatest job payoffs. Then, starting from the obtained partial schedule, add as many of the remaining unscheduled jobs as possible, that can complete into $I_2 =]D_1, D_2]$. And so on for each interval I_k, while there still are unscheduled jobs.

Each computed subschedule of jobs completing in $]D_{k-1}, D_k]$, $k = 1, \ldots, K$, is a right-shifted schedule completing at D_k, by construction with SDD-algorithm. This subschedule is left-shifted before computing the next subschedule, in order to allow as many time units as possible to be available for a possible straddling job, starting before D_k and completing after D_k, in the next subschedule. This left-shift is performed while maintaining feasibility (cf. the definition of a left-shifted schedule, p. 26), and does not decrease the payoffs of the left-shifted jobs. Figures 3.11 and 3.12 illustrate the constructive algorithm.

The formal pseudocode is described in Algorithm 2 (p. 42), and makes use of SDD-algorithm (described in Chapter 2, p. 29). For any feasible (partial) schedule S, we denote by $\mathsf{left\text{-}shift}(S)$ the left-shifted (partial) schedule that schedules the jobs of S in the same order (cf. the definition of a left-shifted schedule, p. 26).

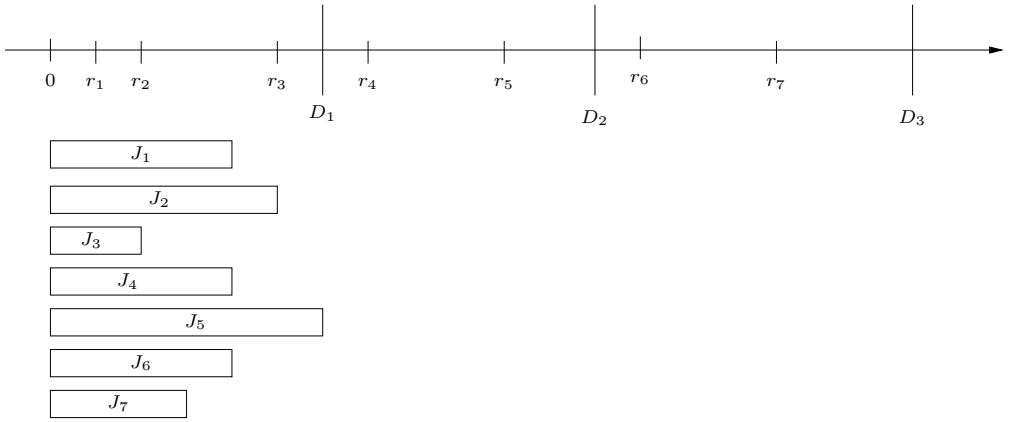


Figure 3.11: An instance of $1|r_i|\sum V_k$ with $N = 7$ jobs.

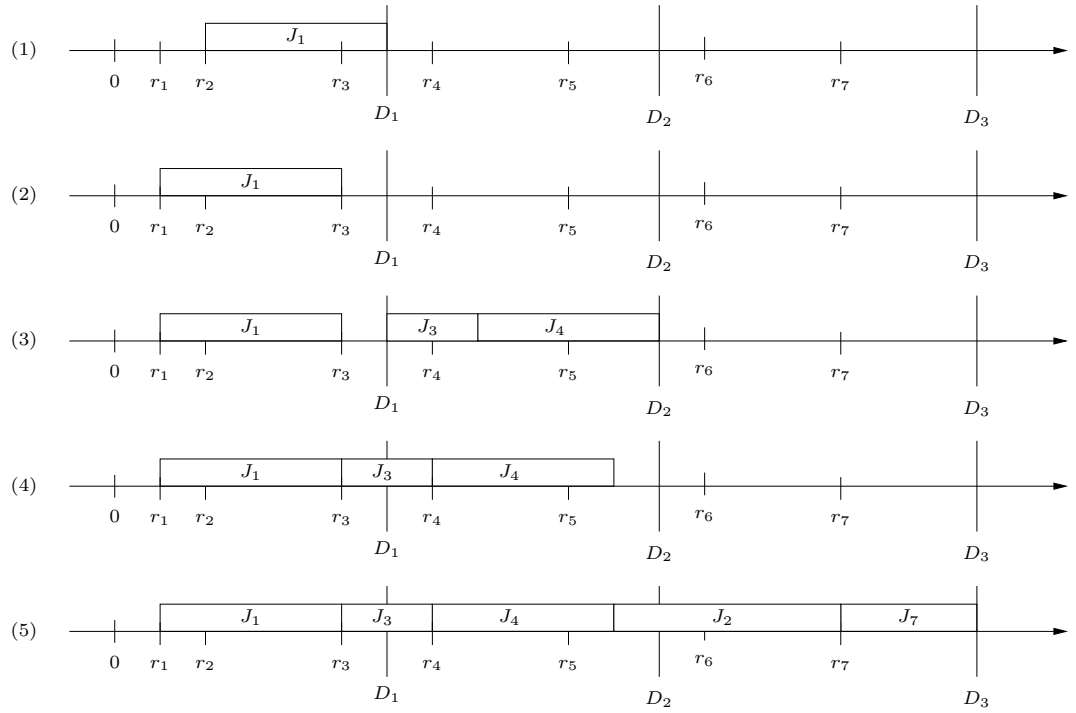


Figure 3.12: The construction of a feasible solution for the instance of Figure 3.11, in five steps: (1) $SDD(\mathcal{J}^{all}, 0, D_1)$;
(2) Left-shifting J_1;
(3) $SDD(\{J_2, J_3, J_4, J_5, J_6, J_7\}, C_1, D_2)$;
(4) Left-shifting S_2;
(5) $SDD(\{J_2, J_5, J_6, J_7\}, C_{max}(S_2), D_3)$.

Input: $\mathcal{J}^{all}$, $D_1, \ldots, D_K$
Output: S^l, l_bound
1 $\mathcal{J} \leftarrow \mathcal{J}^{all}$
2 $S_1^l \leftarrow SDD(\mathcal{J}, 0, D_1)$
3 $S_1^l \leftarrow$ left-shift(S_1^l)
4 $\mathcal{J} \leftarrow \mathcal{J} \backslash \mathcal{J}(S_1^l)$
5 **for** $k = 2$ ***to*** K **do**
6 $\quad S_k^l \leftarrow SDD(\mathcal{J}, C_{max}(S_{k-1}^l), D_k)$
7 $\quad S_k^l \leftarrow$ left-shift(S_k^l)
8 $\quad \mathcal{J} \leftarrow \mathcal{J} \backslash \mathcal{J}(S_k^l)$
9 $S_{K+1}^l \leftarrow$ schedule obtained by scheduling the jobs of $\mathcal{J}$ in ERD order after D_K
10 **return** $S_1^l.S_2^l.\ldots.S_{K+1}^l$, $\sum_{k=1}^{K} k|\mathcal{J}(S_{K-k+1}^l)|$

Algorithm 2: Computation of a lower bound on the optimal payoff of $1|r_i|\sum V_k$.

Proposition 4. *Algorithm 2 produces a feasible schedule for* $1|r_i|\sum V_k$.

Proof. Since the SDD-algorithm yields a feasible subschedule, the subschedules constructed at lines 2 and 6 are feasible. The left-shift operations of lines 3 and 7 are performed while maintaining feasibility. Moreover, the subschedules do not overlap, since the SDD-algorithm is used on intervals where no jobs are scheduled yet (line 2 and line 6). Additionally, there are no jobs scheduled in more than one subschedule, since $\mathcal{J}$ is updated at lines 4 and 8. Finally, scheduling the remaining jobs after D_K maintains feasibility and enables to have a complete schedule. Therefore, S^l is feasible. □

Algorithm 2 uses SDD-algorithm (whose complexity is $O(NlogN)$) K times, hence its complexity is $O(KNlogN)$. From Proposition 4, Algorithm 2 returns a lower bound on the optimal payoff of $1|r_i|\sum V_k$.

We show next how to compute upper bounds. In the Branch and Bound algorithm, upper bounds are mainly computed starting from partial schedules. However, for the sake of clarity, we first describe how to compute an initial upper bound.

3.3.2 Initial upper bound

By Property 1 (p. 31), the SDD-algorithm applied on $\mathcal{J}^{all}$ over any interval $[0, D_k]$, $k = 1, \ldots, K$, yields a schedule with the maximal number $N(\mathcal{J}^{all}, 0, D_k)$ of jobs that can complete at or before D_k in a feasible schedule.

NOTATIONS. We denote $N(\mathcal{J}^{all}, 0, D_k)$ by U_k.
We denote by S^{D_k} the schedule obtained with $SDD(\mathcal{J}^{all}, 0, D_k)$, $k = 1, \ldots, K$.
Hence, $U_k = |\mathcal{J}(S^{D_k})|$.

An upper bound on the optimal payoff of $1|r_i|\sum V_k$ is obtained by considering the payoff of a hypothetical schedule where U_k jobs complete at or before D_k, for each

$k = 1, \ldots, K$. Algorithm 3 computes such an upper bound.

Input: $\mathcal{J}^{all}$, $D_1, \ldots, D_K$
Output: *u_bound*
1 $u\_bound \leftarrow 0$
2 **for** $k = 1$ ***to*** K **do**
3 $\quad S^{D_k} \leftarrow SDD(\mathcal{J}^{all}, 0, D_k)$
4 $\quad U_k \leftarrow |S^{D_k}|$
5 $\quad u\_bound \leftarrow u\_bound + U_k$
6 **end**
7 **return** *u_bound*

Algorithm 3: Computation of an upper bound on the optimal payoff of $1|r_i|\sum V_k$.

Algorithm 3 uses SDD-algorithm K times, hence its complexity is $O(KNlogN)$.

Proposition 5. *Algorithm 3 returns an upper bound on the optimal payoff* $\sum V_k$ *for* $1|r_i|\sum V_k$.

Proof. Let N_k^* be the number of jobs completing in $[0, D_k]$, $k = 1, \ldots, K$, in an optimal schedule S^*. Algorithm 3 computes U_k as the number of jobs of $SDD(\mathcal{J}^{all}, 0, D_k)$, which is the maximal number of jobs that can complete in $[0, D_k]$, $k = 1, \ldots, K$, in any feasible schedule. Therefore, $N_k^* \leq U_k$, for every $k = 1, \ldots, K$. Hence, $v(S^*) = \sum_{k=1}^{K} N_k^* \leq \sum_{k=1}^{K} U_k = u\_bound$, where *u_bound* is the value computed by Algorithm 3. □

3.3.3 Upper bound for a partial schedule

We call upper bound on node n^a an upper bound on the payoff of the schedules that can be obtained by completing the partial schedule S^a associated to n^a. The rationale of the computation of an upper bound on a node n^a is similar to that on the root node (initial upper bound).

NOTATION. The variable $U_k(S^a)$ (U_k when no ambiguity is possible), $k = 1, \ldots, K$, represents an upper bound on the number of jobs that can be inserted into S^a, while completing at or before D_k.

Hence, an upper bound computed at node n^a is $u\_bound(S^a) = \sum_{k=1}^{K} U_k(S^a)$. Therefore, the difference with the initial upper bound (Subsection 3.3.2) stands in the way the U_k values are computed.

NOTATIONS. Given a partial schedule S^a that satisfies the two structural properties of Propositions 2 and 3 (p. 34), recall that $B(S_k^a)$ (resp. $C_{max}(S_k^a)$) denotes the starting time (resp. the completion time) of the k-th block S_k^a, $k = 1, \ldots, K$.

Moreover, the variable L_k, $k = 2, \ldots, K$, represents the number of time units that are unavailable due to block S_k^a (cf. Figure 3.13), i.e. the time units used to process the jobs of S_k^a, and also the time units preceding S_k^a in the interval $I_k =]D_{k-1}, D_k]$. Indeed,

since we only consider ERD-schedules, the time units preceding S_k^a in I_k cannot be used by the Branch and Bound algorithm to insert jobs into S^a (cf. Branching rule, p. 36). More formally, we have the following notation.

NOTATION. We set: $L_k = C_{max}(S_k^a) - \min(B(S_k^a), D_{k-1}), k = 2, \ldots, K$ (see Figure 3.13).

The block S_{K+1}^a is not considered for the computation of an upper bound, since its jobs are worth 0 for the payoff (moreover, notice that we can assume that S_{K+1}^a always starts after D_K).

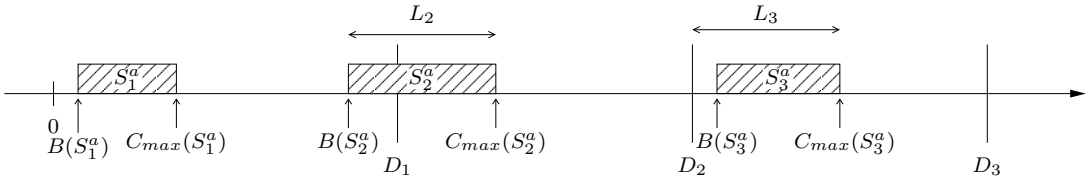


Figure 3.13: A partial schedule S^a of a node n^a of the Branch and Bound tree.

Given a partial schedule S^a, we denote by $\Delta(S_k^a)$, $k = 1, \ldots, K$, the value of the maximal possible right-shift of S_k^a that maintains the same payoff for each job, i.e. such that all the jobs of $S_{k'}^a$, $k \leq k' < K+1$, still complete in $I_{k'}$ after the right-shift.

NOTATION. The maximal possible right-shift of S_K^a is $\Delta(S_K^a) = D_K - C_{max}(S_K^a)$. The maximal possible right-shift of S_k^a, $k = 1, \ldots, K-1$, is: $\Delta(S_k^a) = \min\{D_k - C_{max}(S_k^a), B(S_{k+1}^a) - C_{max}(S_k^a) + \Delta(S_{k+1}^a)\}$ (see Figure 3.14).

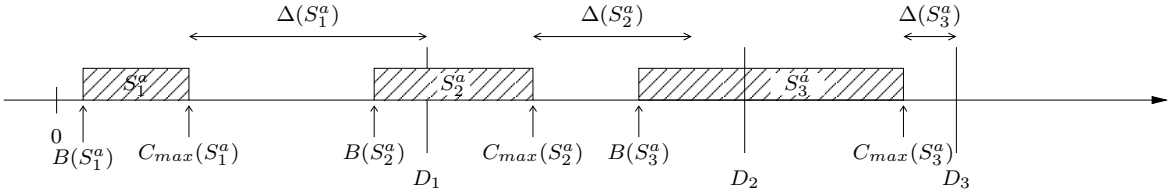


Figure 3.14: Illustration of Δ values. $\Delta(S_3^a) = D_3 - C_{max}(S_3^a)$; $\Delta(S_2^a) = B(S_3^a) - C_{max}(S_2^a) + \Delta(S_3^a)$; $\Delta(S_1^a) = D_1 - C_{max}(S_1^a)$.

To get an upper bound on the partial schedule S^a of Figure 3.13, we need to compute the three values $U_1(S^a)$, $U_2(S^a)$ and $U_3(S^a)$ (U_1, U_2 and U_3, for shortness).

In a general way, a partial schedule S^a of depth e contains the jobs $J_1, \ldots, J_e$. Consequently, the only jobs that will be added to S^a in the further steps of the Branch and Bound algorithm are the jobs of $E = \{J_{e+1}, \ldots, J_N\}$. In order to maintain the ERD-schedule structure, these jobs can only complete between $C_{max}(S_k^a)$ and D_k, $k = 1, \ldots, K$, (possibly on an interval strictly included in $[C_{max}(S_k^a), D_k]$, depending on the

already scheduled blocks), or after D_K. In order to compute an upper bound on the payoffs of the descendants of n^a, we will insert the jobs of E into S^a while allowing a so called block-preemption (defined later in Definition 5).

The jobs of E will only be inserted in some "employable intervals", which guarantee that the ERD-schedule structure is maintained and that we obtain an upper bound. These employable intervals depend on the current computed U_k. We illustrate on Figure 3.15 the employable intervals for U_3 with bold lines.

When computing a given U_k, the considered employable intervals are $[C_{max}(S^a_{k'}), \min(D_{k'}, B(S^a_{k'+1}))]_{k'=1,\ldots,k-1}$ and $[C_{max}(S^a_k), \min\{D_k, B(S^a_{k+1}) + \Delta(S^a_{k+1})\}]$.

Let us show, on the example of Figure 3.15, how to compute U_3. To compute an upper bound on the maximal number of jobs of E that can be scheduled in the bold intervals, we will ignore the already scheduled blocks, and consider a "shrunk" horizon composed of the concatenated bold intervals. Indeed, in a similar way as for the computation of an initial upper bound, the use of SDD-algorithm is necessary to compute each value $U_k(S^a)$. And it appears that it is easier to use SDD-algorithm on an uninterrupted horizon, than to design a variant of SDD-algorithm that deals with the blocks of S^a.

The shrunk horizon is obtained by defining some alternative delivery dates, as shown in Figure 3.16. We have: $D'_3 = D_3$, $D'_2 = C_{max}(S^a_3)$, $D'_1 = C_{max}(S^a_2) + L_3$ and $D'_0 = C_{max}(S^a_1) + L_2 + L_3$. In a general way, $D'_K = D_K$, and $D'_k = C_{max}(S^a_{k+1}) + \sum_{j=k+2}^{K} L_j$, for all $k \in \{0, \ldots, K-1\}$.

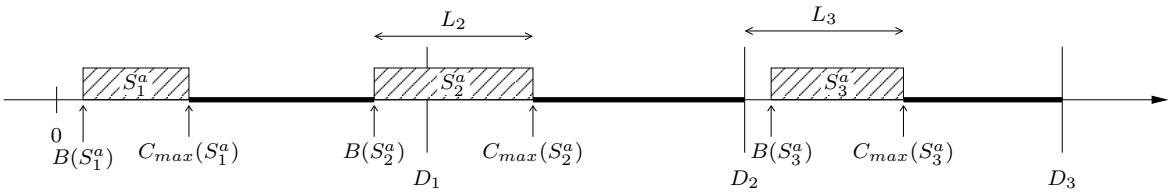


Figure 3.15: The employable intervals for U_3.

Definition 5. *Let S^a be a partial schedule and E the set of jobs that will be added to S^a in the further steps of the Branch and Bound algorithm. We say that block-preemption is allowed on the jobs of E when interrupting the execution of a job is possible only at the end of a bold interval, and if the same job is the first to be processed at the beginning of the next bold interval (see Figure 3.17).*

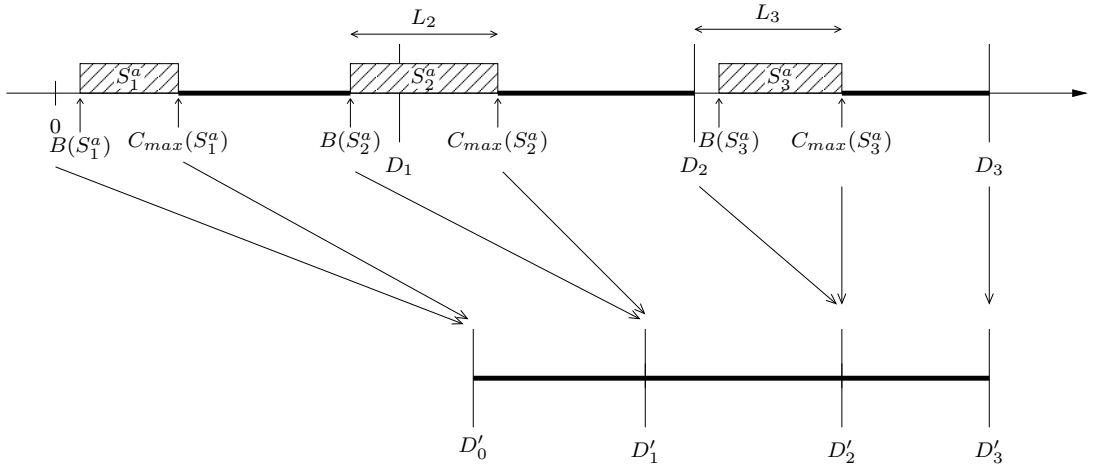


Figure 3.16: The "shrunk" horizon with the alternative delivery dates.

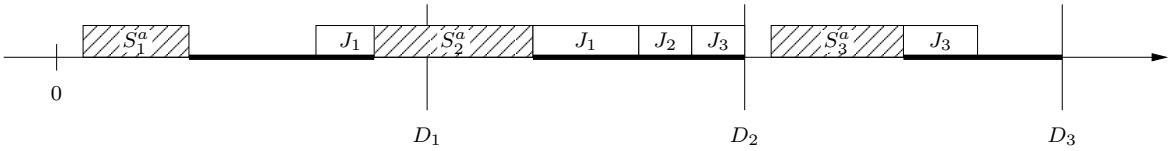


Figure 3.17: On this example, a partial schedule S^a is represented by its striped blocks S_1^a, S_2^a, S_3^a, and the jobs of $E = J_1, J_2, J_3$ are inserted into S^a while observing block-preemption (J_1, J_3).

On the yielded shrunk horizon, we can use SDD-algorithm. We obtain a partial schedule $\tau^{D'_3}(S^a)$ of the jobs of E on the shrunk horizon $[D'_0, D'_3]$. $\tau^{D'_3}(S^a)$ can be converted into a partial schedule $\sigma^{D_3}(S^a)$ on the original horizon. $\sigma^{D_3}(S^a)$ schedules the jobs of E on the bold intervals, while allowing block-preemption. Block-preemption ensures that $\sigma^{D_3}(S^a)$ provides an upper bound. The conversion from $\tau^{D'_3}(S^a)$ to $\sigma^{D_3}(S^a)$ is simply made by separating the bold intervals that were stuck together in the shrunk horizon, so that they are again integrated in the original horizon, while modifying the starting times of the jobs accordingly. An example can be seen in Figure 3.18.

We have already seen how to adapt the delivery dates, in order to define the new horizon. The other parameters that need to be updated on the new horizon are the release dates of the jobs of E. Notice that, on the original horizon, the release dates r of the jobs of E can all be replaced by equivalent release dates $\tilde{r}$ on the bold intervals, in the following way (cf. Figure 3.19, where $E = \{J_f, J_g, J_h\}$). When a release date r_i is on a bold interval, it remains unchanged. Otherwise, the equivalent release date $\tilde{r}_i$ is equal to the next completion time of a block: $\tilde{r}_i = \min_{k=1,\dots,K}\{C_{max}(S_k^a) | C_{max}(S_k^a) > r_i\}$. These new release dates are equivalent to the original ones, since we only consider scheduling the jobs of E on the bold intervals.

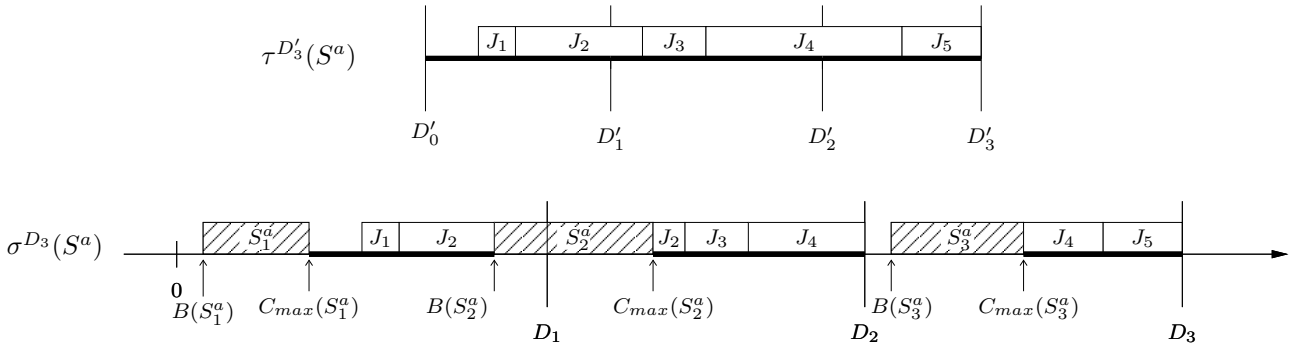


Figure 3.18: The solution $\tau^{D'_3}(S^a)$ is converted in a solution $\sigma^{D_3}(S^a)$.

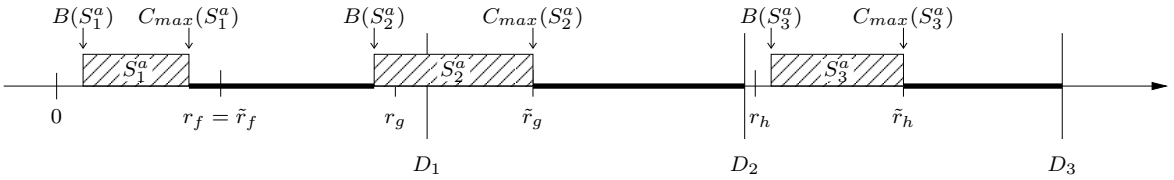


Figure 3.19: The equivalent release dates $\tilde{r}$ of the jobs of $E = J_f, J_g, J_h$.

This way, when the horizon is shrunk, the alternative release dates r' of the jobs of E appear in the shrunk horizon. Their values are obtained by translating the $\tilde{r}$ release dates in a similar way as for the alternative delivery dates, as follows (cf. Figure 3.20). Let D_{k_i} be the smallest delivery date such that $\tilde{r}_i < D_{k_i}$. Then, $r'_i = \tilde{r}_i + \sum_{j=k_i+1}^{K} L_j$. On the same example as above, in Figure 3.20 we have: $r'_h = \tilde{r}_h$, $r'_g = \tilde{r}_g + L_3$ and $r'_f = \tilde{r}_f + L_2 + L_3$.

We have seen how to build a shrunk horizon for the computation of U_3 on an example. On the same example, U_2 is computed on the same shrunk horizon, by only considering it between D'_0 and D'_2 (see Figure 3.21).

As for U_1, we need a supplementary step to deduce the available shrunk horizon. Indeed $B(S^a_2) < D_1$ (see for instance Figure 3.19), thus to ensure that we really calculate an upper bound on the jobs that can complete before D_1, we need to right-shift S^a_2 as much as possible, i.e. of $\Delta(S^a_2)$ time units.

In a general way, right-shifting S^a_{k+1} of $\Delta(S^a_{k+1})$ units does not change the structure of the partial solution, since S^a_{k+1} still completes at or before D_{k+1}. However, this right-shift allows the use of the freed time units before D_k to block-preemptively schedule the jobs of E, in order to compute U_k, which ensures that the result yields an upper bound. Notice that, for our purposes, S^a_{k+1} needs only be shifted of $\min\{max(0, D_k - B(S^a_{k+1})), \Delta(S^a_{k+1})\}$ time units.

NOTATION. $\delta(S^a_{k+1}) = \min\{max(0, D_k - B(S^a_{k+1})), \Delta(S^a_{k+1})\}$, $k = 1, \ldots, K$.

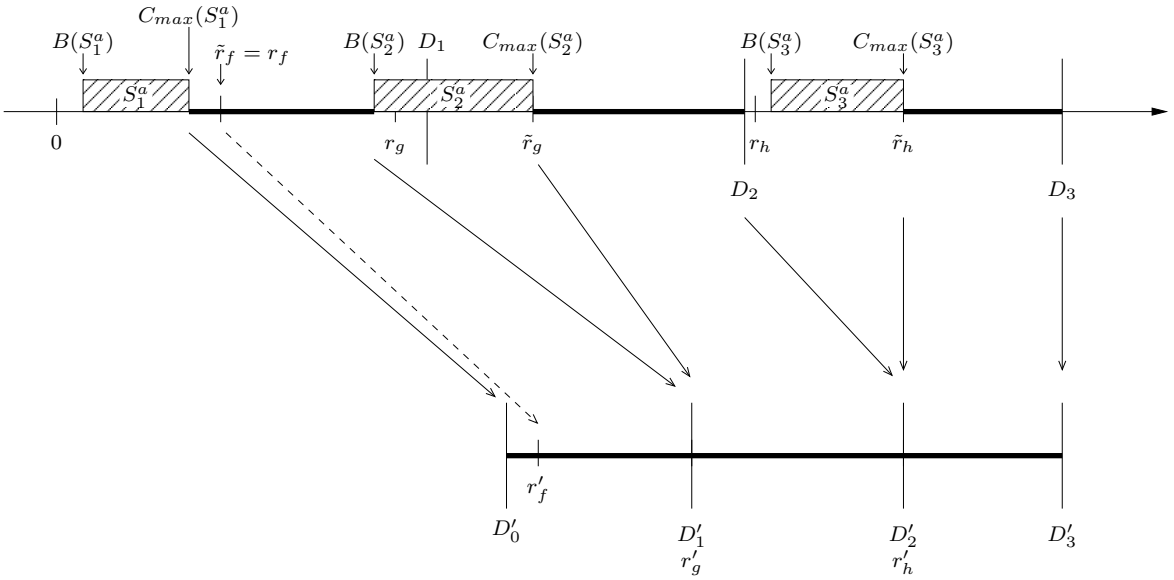


Figure 3.20: The shrunk horizon with the alternative delivery and release dates.

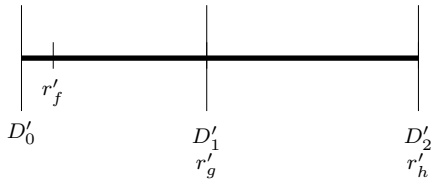


Figure 3.21: The shrunk horizon for U_2.

To come back to the example, you can see on Figure 3.22 the right-shift of S_2.

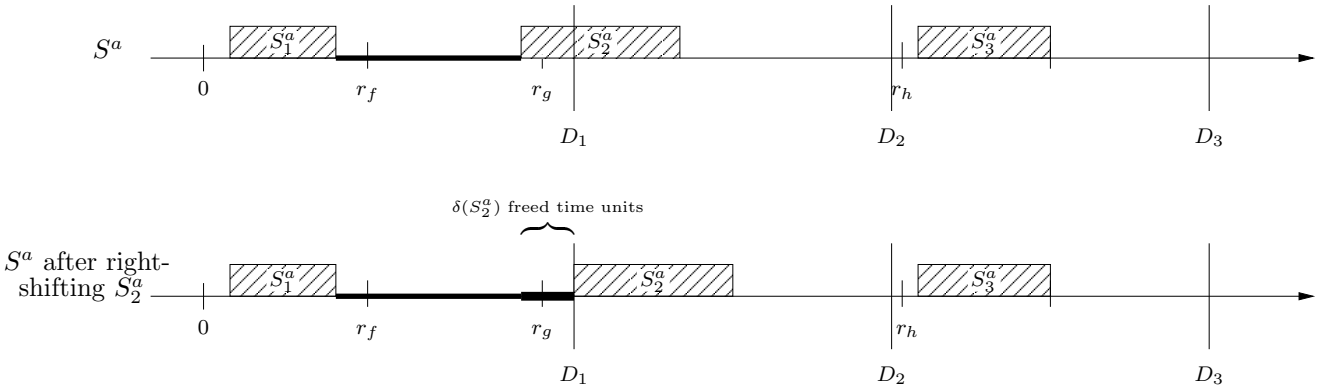


Figure 3.22: The right-shift of S_2^a.

Therefore, to obtain the shrunk horizon for U_1 from the shrunk horizon for U_3, we

need to add the interval freed by the shift (see Figure 3.23) to $[D'_0, D'_1]$. Hence, D'_1 must be recalculated: $D'_1(U_1) = D'_1(U_3) + \delta(S^a_2)$. Moreover, for the jobs whose original release dates are on the "freed time units interval" (see Figure 3.22), we need to update their alternative release dates: on the example, $r'_g = D'_1(U_1) - (D_1 - r_g)$.

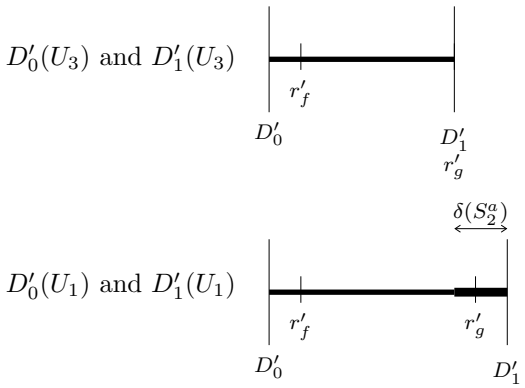


Figure 3.23: The shrunk horizon for U_1.

The procedure that yields an upper bound on n^a does first compute U_K by the means of a shrunk horizon. Then, this shrunk horizon is updated for the computation of each of the U_k values, $k = K-1, \ldots, 1$. These operations are formalized in the following. First, Algorithm 4 (p.50) computes the horizon for U_K. Then, given this shrunk horizon, Algorithm 5 (p.51) computes every U_k, $k = K, \ldots, 1$, and returns the upper bound. Moreover, Algorithm 5 uses the CONV-algorithm (described by Algorithm 6, p. 52) to convert a schedule $\tau^{D'_k}$ on the shrunk horizon into a block-preempted schedule σ^{D_k} on the original horizon.

NOTATION. We denote by $E_{r'}$ the set of jobs of E where alternative release dates replace the original ones.

Proposition 6. *Given a partial schedule S^a, Algorithm 5 returns an upper bound.*

Proof. The general principle for the computation of an upper bound on n^a, consisting in summing the U_k, is the same as for an initial upper bound, and is performed at lines 18-19 of Algorithm 5. It remains to prove that U_k, as computed by Algorithm 5, is an upper bound on the number of jobs that can be added to partial schedule S^a between 0 and D_k. Recall that, for each $k \in \{1, \ldots, K\}$, we call employable intervals $[C_{max}(S^a_{k'}), \min(D_{k'}, B(S^a_{k'+1}))]_{k'=1,\ldots,k-1}$ and $[C_{max}(S^a_k), \min\{D_k, B(S^a_{k+1}) + \Delta(S^a_{k+1})\}]$. First, we show that, for any $k \in \{1, \ldots, K\}$, the maximal number of jobs of E that can be scheduled on the employable intervals when block-preemption is allowed, is an upper bound on the number of jobs that can be added to partial schedule S^a between 0 and D_k. Second, we show that, for each $k \in \{1, \ldots, K\}$, U_k, as computed by Algorithm 5,

Input: e, $r_{e+1}, \ldots, r_N$, $D_0, \ldots, D_K$, S^a
Output: $r'_{e+1}, \ldots, r'_N$, $D'_0, \ldots, D'_K$

$D'_K \leftarrow D_K$
$k \leftarrow K-1$
$i \leftarrow N$
$translate \leftarrow 0$
while $i \geq e+1$ **do**
  if $r_i > \min(D_k, B(S^a_{k+1}))$ **then**
    if $r_i < C_{max}(S^a_{k+1})$ **then**
      $\tilde{r}_i \leftarrow C_{max}(S^a_{k+1})$
    else
      $\tilde{r}_i \leftarrow r_i$
    $r'_i \leftarrow \tilde{r}_i + translate$
    $i \leftarrow i-1$
  else
    $L_{k+1} \leftarrow C_{max}(S^a_{k+1}) - \min(D_k, B(S^a_{k+1}))$
    $translate \leftarrow translate + L_{k+1}$
    $D'_k \leftarrow D_k + translate$
    $k \leftarrow k-1$
for $j = k$ ***to*** 1 **do**
  $L_{j+1} \leftarrow C_{max}(S^a_{j+1}) - \min(D_j, B(S^a_{j+1}))$
  $translate \leftarrow translate + L_{j+1}$
  $D'_j \leftarrow D_j + translate$
  $j \leftarrow j-1$
return $r'_{e+1}, \ldots, r'_N$, $D'_0, \ldots, D'_K$

Algorithm 4: Computation of the shrunk horizon for U_K, starting from a partial schedule S^a.

is the maximal number of jobs of E that can be scheduled on the employable intervals when block-preemption is allowed.

Given a node n^a of depth e in the branching tree, the jobs that can be added to S^a in the Branch and Bound are the jobs of $E = \{J_{e+1}, \ldots, J_N\}$. In order to maintain the ERD-schedule structure, the jobs of E can only complete after $C_{max}(S^a_k)$ if they complete in I_k. When computing U_k, $k = 1, \ldots, K$, we only consider adding the jobs of E between 0 and D_k. In this case, if the block S^a_{k+1} is straddling, it must be momentarily right-shifted of its maximal shift $\Delta(S^a_{k+1})$, to maximize the length of the total idle time before D_k. Indeed, in the Branch and Bound algorithm, S^a_{k+1} can be right-shifted in order to allow a job to be inserted into I_k. Therefore, by right-shifting S^a_{k+1} of its maximal possible shift, we ensure that the considered set of intervals is not suboptimal. We showed that the last employable interval, when computing U_k, is $[C_{max}(S^a_k), \min\{D_k, B(S^a_{k+1}) + \Delta(S^a_{k+1})\}]$.

Input: E, $r'_{e+1}, \ldots, r'_N$, $D_1, \ldots, D_K$, $D'_0, \ldots, D'_K$, S^a
Output: u_bound, $\tau^{D'_1}, \ldots, \tau^{D'_K}$, $\sigma^{D_1}, \ldots, \sigma^{D_K}$

$\tau^{D'_K} \leftarrow SDD(E_{r'}, D'_0, D'_K)$
$U_K \leftarrow |\tau^{D'_K}|$
$u\_bound \leftarrow u\_bound + U_K$
$\Delta_K \leftarrow D_K - C_{max}(S^a_K)$
$k \leftarrow K - 1$
while $k \geq 1$ ***and*** $r_{e+1} < D_k$ **do**
  $\delta_{k+1} \leftarrow \min\{max(0, D_k - B(S^a_{k+1})), \Delta_{k+1}\}$
  if $\delta_{k+1} > 0$ **then**
    $D'_k \leftarrow D'_k + \delta_{k+1}$
    for $i = N$ ***to*** $e + 1$ **do**
      if $r_i > B(S^a_{k+1})$ **then**
        if $r_i < B(S^a_{k+1}) + \delta_{k+1}$ **then**
          $r'_i \leftarrow D'_k - (B(S^a_{k+1}) + \delta_{k+1} - r_i)$
      else
        break
  $\tau^{D'_k} \leftarrow SDD(E_{r'}, D'_0, D'_k)$
  $\sigma^{D_k} \leftarrow CONV(\tau^{D'_k}, D'_0, \ldots, D'_K, D_1, \ldots, D_K, S)$
  $U_k \leftarrow |\tau^{D'_k}|$
  $u\_bound \leftarrow u\_bound + U_k$
  $\Delta_k \leftarrow \min\{D_k - C_{max}(S^a_k), B(S^a_{k+1}) - C_{max}(S^a_k) + \Delta_{k+1}\}$
  $k \leftarrow k - 1$
for $i = k$ ***to*** 1 **do**
  $\tau^{D'_k} \leftarrow$empty schedule
return u_bound, $\tau^{D'_1}, \ldots, \tau^{D'_K}$, $\sigma^{D_1}, \ldots, \sigma^{D_K}$

Algorithm 5: Computation of an upper bound on n^a, starting from the outputs of Algorithm 4.

When computing U_k, right-shifting the blocks $S^a_1, \ldots, S^a_k$ yields a suboptimal set of intervals, since it does not increase the number of available time units, though it makes free time units to be earlier, which is suboptimal, since jobs have release dates and block-preemption is allowed. Finally, allowing block-preemption guarantees to produce an upper bound on the number of jobs that can be added non-preemptively to the partial schedule. Indeed, block-preemption is a relaxation, and it allows to use the maximal number of time units to schedule jobs. We showed that for any $k \in \{1, \ldots, K\}$, the maximal number of jobs of E that can be scheduled on the employable intervals defined above, when block-preemption is allowed, is an upper bound on the number of jobs that can be added to partial schedule S^a between 0 and D_k.

Let us show that U_k, as computed by Algorithm 5, is the maximal number of jobs of

Input: $C_{l_1}(\tau^{D'_k}), \ldots, C_{l_{U_k}}(\tau^{D'_k})$, $D'_0, \ldots, D'_K$, $D_1, \ldots, D_K$, S^a
Output: $\sigma^{D_k} = C_{l_1}(\sigma^{D_k}), \ldots, C_{l_{U_k}}(\sigma^{D_k})$

```
1  j ← k − 1
2  translate ← 0
3  i ← U_k
4  while i ≥ 1 do
5      if C_{l_i}(τ^{D'_k}) > D'_j then
6          C_{l_i}(σ^{D_k}) ← C_{l_i}(τ^{D'_k}) − translate
7          i ← i − 1
8      else
9          L_{j+1} ← C_max(S^a_{j+1}) − min(D_j, B(S^a_{j+1}))
10         translate ← translate + L_{j+1}
11         j ← j − 1
12 return σ^{D_k} = C_{l_1}(σ^{D_k}), ..., C_{l_{U_k}}(σ^{D_k})
```

Algorithm 6: The CONV-algorithm produces a block-preempted schedule σ^{D_k}, starting from its corresponding partial schedule $\tau^{D'_k}$ on the shrunk horizon. A schedule is completely represented by the completion times of its jobs.

E that can be scheduled on the employable intervals when block-preemption is allowed. We know, by Property 1 (p. 31), that SDD-algorithm schedules the maximal number of jobs into $[D'_0, D'_k]$ (line 15 of Algorithm 5). Therefore, it remains to show that an optimal sequence on $[D'_0, D'_k]$ is also an optimal sequence on the employable intervals of $[0, D_k]$.

Given an optimal schedule $\tau^{D'_k}$ on $[D'_0, D'_k]$ (computed at line 16), consider each "portion" of $\tau^{D'_k}$ scheduled between two consecutive alternative delivery dates. The jobs of $\tau^{D'_k}$ that are straddling on the delivery dates are scheduled partly in different portions. By scheduling each portion of $\tau^{D'_k}$ in the corresponding employable interval of $[0, D_k]$ (you can refer to Figure 3.18 for an example), we obtain a "block-preempted" schedule σ^{D_k} that schedules the jobs of E on the employable intervals of $[0, D_k]$. This operation is executed by CONV-algorithm at line 17 of Algorithm 5. σ^{D_k} satisfies the release dates, since, if we translate the alternative release dates r' with the portions of $\tau^{D'_k}$, we obtain the release dates $\tilde{r}$ on the employable intervals, which are equivalent to the original release dates r, when the only allowed intervals to schedule jobs are the employable intervals of $[0, D_k]$. It remains to show that σ^{D_k} is optimal, i.e. contains the maximal number of jobs that can be scheduled on the employable intervals of $[0, D_k]$, when block-preemption is allowed. By contradiction, suppose that there exists a block-preempted schedule $\widehat{\sigma}^k$ of jobs of E on the employable intervals, such that $|\mathcal{J}(\widehat{\sigma}^k)| > |\mathcal{J}(\sigma^{D_k})|$. Therefore, by the inverse operation used to transform $\tau^{D'_k}$ into σ^{D_k}, we obtain from $\widehat{\sigma}^k$ a solution $\widehat{\tau}^k$ on $[D'_0, D'_k]$. $\widehat{\tau}^k$ is feasible, since the alternative release dates on the shrunk horizon can be obtained by translating the release dates with the portions of σ^{D_k}. Since

$|\mathcal{J}(\widehat{\tau}^k)| > |\mathcal{J}(\tau^{D'_k})|$, this leads to a contradiction on Property 1 (p.31) on $\tau^{D'_k}$. □

Proposition 7. *The time complexity for the computation from scratch of an upper bound at a node n^a is $O(KNlogN)$.*

Proof. The computation of an upper bound needs the execution of Algorithm 4 followed by Algorithm 5. Algorithm 4 is executed in $O(N+K)$ time, since the while loop of lines 5-17 is executed at most N times, while the for loop of lines 18-22 is executed at most K times. Algorithm 5 is executed in $O(KNlogN)$ time, since the while loop of lines 6-21 is executed at most K times; and in each of these loops, the for loop of lines 10-15 is executed at most N times, SDD-algorithm is executed once, in time $O(NlogN)$, and CONV-algorithm is executed once, in time $O(N)$. □

Justification of the branching structure. If we had chosen a classic branching scheme where jobs are scheduled sequentially, we would not have needed deal with employable intervals. In this case, by denoting by $C_{max}(S^a)$ the completion time of the last job of the current left-shifted partial schedule S^a, computing an upper bound on a node could have been done by simply applying SDD-algorithm on the intervals $[C_{max}(S^a), D_k]$, forall $\{k \in \{1,\ldots,K\} | D_k > C_{max}(S^a)\}$. Even if in practice this bound computation is faster than that presented in Section 3.3.3, its complexity in the worst case is still $O(KNlogN)$.

On the other side, with the branching rule we chose, we explore at most K^N nodes, because of the dominance of ERD-schedules (cf. Proposition 2, p. 34). With a classic branching scheme, the maximal number of nodes is N^N, which is much larger, as N is intended to be much larger than K.

3.3.3.1 Upper bound computation from the father node bound

In order to avoid redundant computations, we identified some cases where the upper bound on a node n^a can be computed starting from the upper bound on its father node $n^{f(a)}$. In the other cases, this is not possible, and the bound must thus be computed from scratch.

There are three cases where the computation of an upper bound at node n^a is done in $O(1)$, starting from the upper bound at its father node. We also present the rationale for another possible case, in which the upper bound at node n^a is computed in $O(kNlogN)$ time, where D_k is the delivery date associated with n^a, and where in practice we consider fewer jobs than if the bound was computed from scratch.

Let e be the depth of n^a. Then, S^a is obtained from $S^{f(a)}$ by adding J_e to $S^{f(a)}$, as seen in Section 3.2. The value $U_k(S^{f(a)})$, $k = 1,\ldots,K$, has been previously obtained as the number of jobs scheduled in $\tau^{D'_k}(S^{f(a)})$ (as seen in Section 3.3.3).

In order to compute $U_k(S^a)$, we construct $\tau^{D'_k}(S^a)$, $k = 1,\ldots,K$. First, Algorithm 7 computes the shrunk horizon for $U_K(S^a)$, starting from S^a, $S^{f(a)}$ and the shrunk horizon

Input: E, $r'_{e+1}(S^{f(a)}), \ldots, r'_N(S^{f(a)})$, $D_0, \ldots, D_K$, S^a, $S^{f(a)}$
Output: $r'_{e+1}(S^a), \ldots, r'_N(S^a)$, $D'_0, \ldots, D'_K$

$D'_K \leftarrow D_K$
$k \leftarrow K-1$
$i \leftarrow N$
$translate \leftarrow 0$
if $C_{max}(S^{f(a)}_{k+1}) = C_{max}(S^a_{k+1})$ **then** $flag \leftarrow true$
else $flag \leftarrow false$
while $i \geq e+1$ **do**
  if $r_i > \min(D_k, B(S^a_{k+1}))$ **then**
    if $flag$ **then** $r'_i(S^a) \leftarrow r'_i(S^{f(a)})$
    else
      if $r_i < C_{max}(S^a_{k+1})$ **then** $\tilde{r}_i \leftarrow C_{max}(S^a_{k+1})$
      else $\tilde{r}_i \leftarrow r_i$
      $r'_i \leftarrow \tilde{r}_i + translate$
    $i \leftarrow i-1$
  else
    $L_{k+1} \leftarrow C_{max}(S^a_{k+1}) - \min(D_k, B(S^a_{k+1}))$
    $translate \leftarrow translate + L_{k+1}$
    $D'_k \leftarrow D_k + translate$
    $k \leftarrow k-1$
    if $C_{max}(S^{f(a)}_{k+1}) \neq C_{max}(S^a_{k+1})$ **then** $flag \leftarrow false$
for $j = k$ ***to*** 1 **do**
  $L_{j+1} \leftarrow C_{max}(S^a_{j+1}) - \min(D_j, B(S^a_{j+1}))$
  $translate \leftarrow translate + L_{j+1}$
  $D'_j \leftarrow D_j +$
  $j \leftarrow j-1$
return $r'_{e+1}(S^a), \ldots, r'_N(S^a)$, $D'_0, \ldots, D'_K$

Algorithm 7: Computation of the shrunk horizon for $U_K(S^a)$, starting from S^a, $S^{f(a)}$ and the shrunk horizon for $U_K(S^{f(a)})$.

for $U_K(S^{f(a)})$. The complexity of Algorithm 7 is $O(KNlogN)$, but in practice we consider fewer jobs (only from J_{e+1} to J_N). Then, each schedule $\tau^{D'_k}(S^a)$, $k = 1, \ldots, K$, is obtained from $\tau^{D'_k}(S^{f(a)})$ if possible (the possible cases are detailed below), otherwise it is computed from scratch as seen in Section 3.3.3. We denote by $r'_i(S^a)$ (resp. $r'_i(S^{f(a)})$) the alternative release date of job J_i in the shrunk horizon for $U_K(S^a)$ (resp. $U_K(S^{f(a)})$).

The cases where $\tau^{D'_k}(S^a)$ can be derived from $\tau^{D'_k}(S^{f(a)})$ are listed below, with a sketch of proof of their validity.

First, notice that if $r_{e+1} \geq D_k$, then $\tau^{D'_k}(S^a)$ is empty, since no job of $\{J_{e+1}, \ldots, J_N\}$ can complete before D_k (see Figure 3.24). In this case, the bound is obtained in $O(1)$

time.

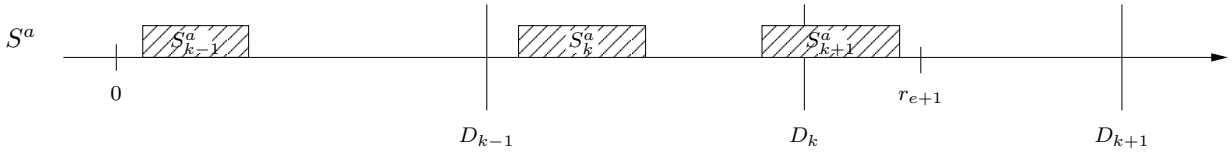


Figure 3.24: Illustration of the case where $r_{e+1} \geq D_k$, on an example where $k = 2$.

In the following cases, we consider that $r_{e+1} < D_k$.

- **Case 1.** (cf. Figure 3.25) Consider the case where:
  - J_e belongs to the schedule $\tau^{D'_k}(S^{f(a)})$: $J_e \in \mathcal{J}(\tau^{D'_k}(S^{f(a)}))$ (which implies that $J_e \in \mathcal{J}(\sigma^{D_k}(S^{f(a)}))$), and
  - the completion time of J_e in $\sigma^{D_k}(S^{f(a)})$ is greater than or equal to the completion time of J_e in S^a: $C_e(\sigma^{D_k}(S^{f(a)})) \geq C_e(S^a)$.

  We have: $\sigma^{D_k}(S^a) = \sigma^{D_k}(S^{f(a)})\backslash J_e$[1].

  Indeed, J_e is the first job of $\sigma^{D_k}(S^{f(a)})$, therefore all the jobs of $\mathcal{J}(\sigma^{D_k}(S^{f(a)}))\backslash\{J_e\}$ can be scheduled between $C_e(\sigma^{D_k}(S^{f(a)}))$ and D_k in addition to S^a which is identical to $S^{f(a)}$ between $C_e(\sigma^{D_k}(S^{f(a)}))$ and D_k. Moreover, no other job can be added to $\sigma^{D_k}(S^a)$ between $C_e(S^a)$ and D_k, otherwise it would have been possible to add it also to $\sigma^{D_k}(S^{f(a)})$.

  In this case, the bound is obtained in $O(1)$ time.

- **Case 2.** (cf. Figure 3.26) Consider the case where:
  - J_e does not belong to the schedule $\tau^{D'_k}(S^{f(a)})$: $J_e \notin \mathcal{J}(\tau^{D'_k}(S^{f(a)}))$,
  - J_e starts at or after D_k in S^a: $C_e(S^a) - p_e \geq D_k$ and
  - $B(S^a_{k+1}) + \delta(S^a_{k+1}) = B(S^{f(a)}_{k+1}) + \delta(S^{f(a)}_{k+1})$.

  We have: $\sigma^{D_k}(S^a) = \sigma^{D_k}(S^{f(a)})$.

  Indeed, since J_e starts after D_k in S^a, the schedule S^a between 0 and D_k is identical to schedule $S^{f(a)}$ between 0 and D_k, except for $B(S^a_{k+1})$ and $B(S^{f(a)}_{k+1})$ that can be different if J_e belongs to S^a_{k+1}. However, since $B(S^a_{k+1}) + \delta(S^a_{k+1}) = B(S^{f(a)}_{k+1}) + \delta(S^{f(a)}_{k+1})$, the same free time units are available between 0 and D_k for both schedules after the right-shifting of S^a_{k+1}, respectively $S^{f(a)}_{k+1}$. Finally, since $J_e \notin \mathcal{J}(\tau^{D'_k}(S^a))$, we deduce that replacing E with $E\backslash\{J_e\}$ as input of SDD-algorithm between 0 and D_k yields the same result.

  In this case, the bound is obtained in $O(1)$ time.

[1] $\sigma^{D_k}(S^{f(a)})\backslash J_e$ is the subschedule of $\sigma^{D_k}(S^{f(a)})$ that includes the jobs of $\mathcal{J}(\sigma^{D_k}(S^{f(a)}))\backslash\{J_e\}$.

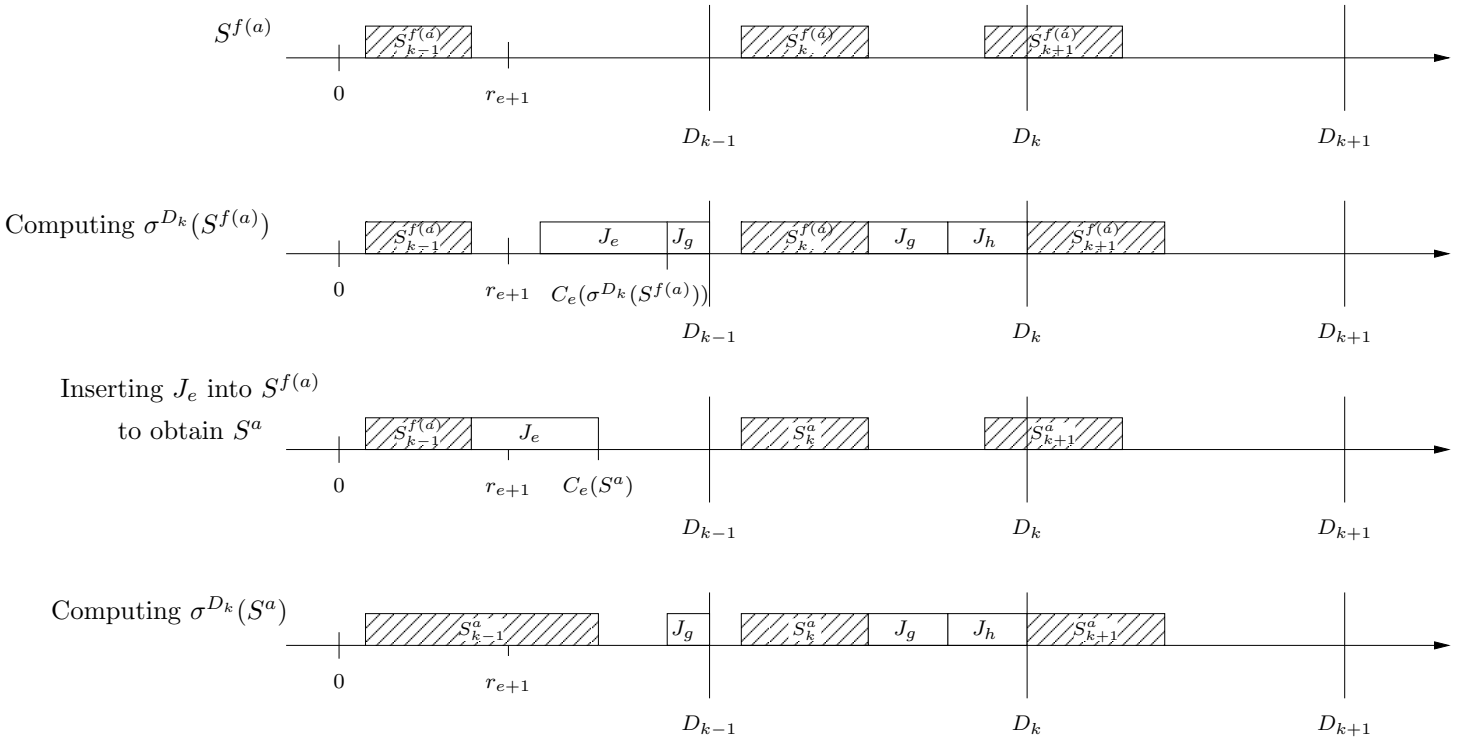


Figure 3.25: Illustration of Case 1, on an example where $k = 2$.

Another possible idea for optimizing the computation of the bound is the following. Suppose that:

- J_e does not belong to the schedule $\tau^{D'_k}(S^{f(a)})$: $J_e \notin \mathcal{J}(\tau^{D'_k}(S^{f(a)}))$ and
- J_e completes before D_k in S^a: $C_e(S^a) < D_k$.

If we compare the shrunk horizons for $U_k(S^a)$ and $U_k(S^{f(a)})$, we notice that there exists a date t_z such that the two shrunk horizons are identical between t_z and D_k, as shown in Figure 3.27.

Then, the rationale of the algorithm to construct $\tau^{D'_k}(S^a)$ starting from $\tau^{D'_k}(S^{f(a)})$ is the following (cf. Figure 3.28). Let J_z be the first job of $\tau^{D'_k}(S^{f(a)})$ starting at or after t_z. Let S^z be the subschedule of $\tau^{D'_k}(S^{f(a)})$ including J_z and the subsequent jobs. Schedule the jobs of S^z, in the same order, on the shrunk horizon for $U_3(S^a)$, in a unique block that completes at $D'_3(S^a)$. We have thus a partial schedule, that can be completed with SDD-algorithm. Indeed, S^z can be considered as the current schedule of SDD-algorithm, after the iteration that considered job J_z. Hence, we can execute the following iterations of SDD-algorithm, considering S^z as the current schedule, and examining each of the jobs of E that precedes J_z in ERD order, in order to add them to the current schedule. The idea is that jobs that had been discarded by SDD-algorithm when constructing $\tau^{D'_k}(S^{f(a)})$ (i.e. the jobs that follow J_z) in ERD order but do not appear in S^z will also

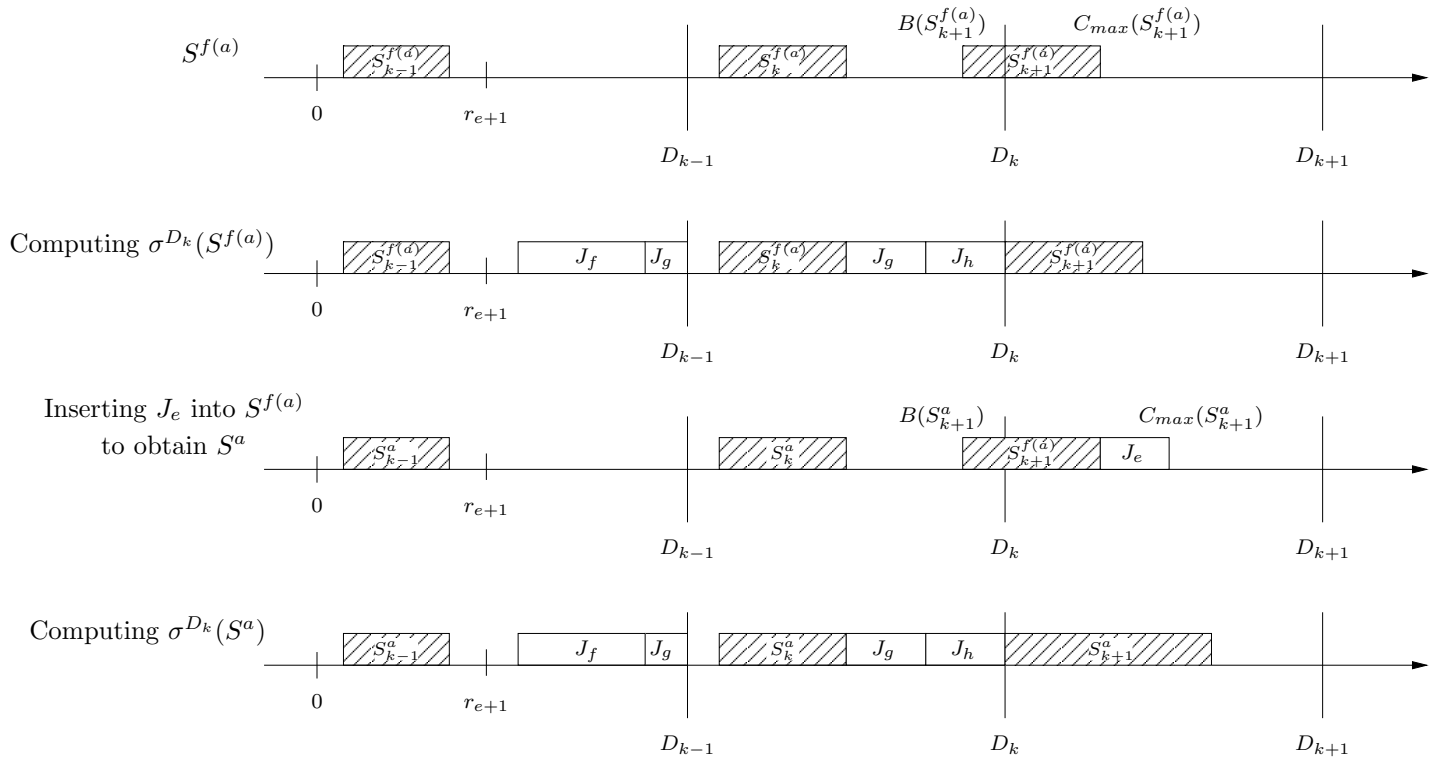


Figure 3.26: Illustration of Case 2, on an example where $k = 2$.

be discarded when constructing $\tau^{D'_k}(S^a)$ from scratch with SDD-algorithm. Indeed, we apply the algorithm with the same set of jobs ($E \backslash \{J_e\}$, since J_e is not in $\tau^{D'_k}(S^{f(a)})$) but on a smaller interval.

3.4 Structural properties for pruning

Let S^a be the current partial schedule, and $E = \{J_{e+1}, \ldots, J_N\}$ the remaining jobs to examine in the Branch and Bound procedure.

Property 5. *If S^a schedules U_K jobs completing at or before D_K, then the only feasible leaf descendant of n^a is the one that schedules all the remaining jobs after D_{K+1}.*

Property 6. *If S^a_K is empty and $r_{e+1} \geq \min(D_{K-2}, B(S^a_{K-1}))$, then the best schedule among those that can be obtained from S^a, schedules exactly $U_{K-1}(S^a) + |\mathcal{J}(S^a_{K-1})|$ jobs completing into I_{K-1}. In this case, every descendant node n^d of n^a satisfying these two conditions is pruned:*

- $r_{e+1} \geq \min(D_{K-1}, B(S^d_K))$, *and*
- S^d *schedules less than* $U_{K-1}(S^a) + |\mathcal{J}(S^a_{K-1})|$.

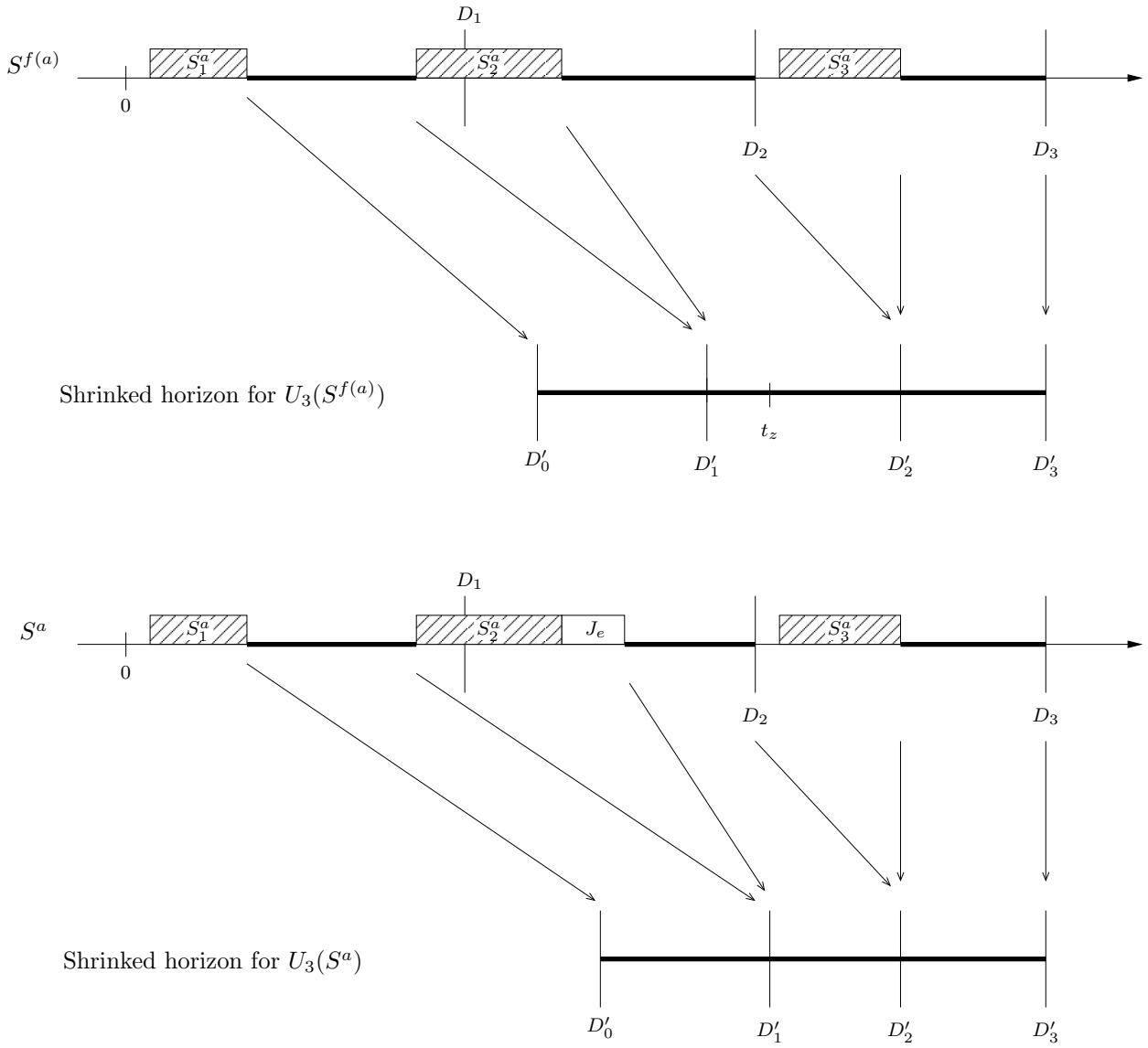


Figure 3.27: The shrunk horizons for $U_k(S^a)$ and $U_k(S^{f(a)})$, on an example where $k = K = 3$. Between $t_z = D'_1(S^a)$ and $D'_3(S^a) = D'_3(S^{f(a)})$, the two horizons are identical.

Property 6 is valid because of the dominance rule for the two delivery dates problem presented in Chapter 4.

Property 7. *If $r_{e+1} \geq \min(D_{K-1}, B(S_K^a))$, then S^a can be completed in polynomial time, with SDD-algorithm.*

Property 8. *For each $k \in \{1, \ldots, K\}$, let J_{i_k} be the first job in the ERD order (i.e. with the smallest index) such that $r_{i_k} + p_{i_k} > D_k$. Then, for each node of the Branch*

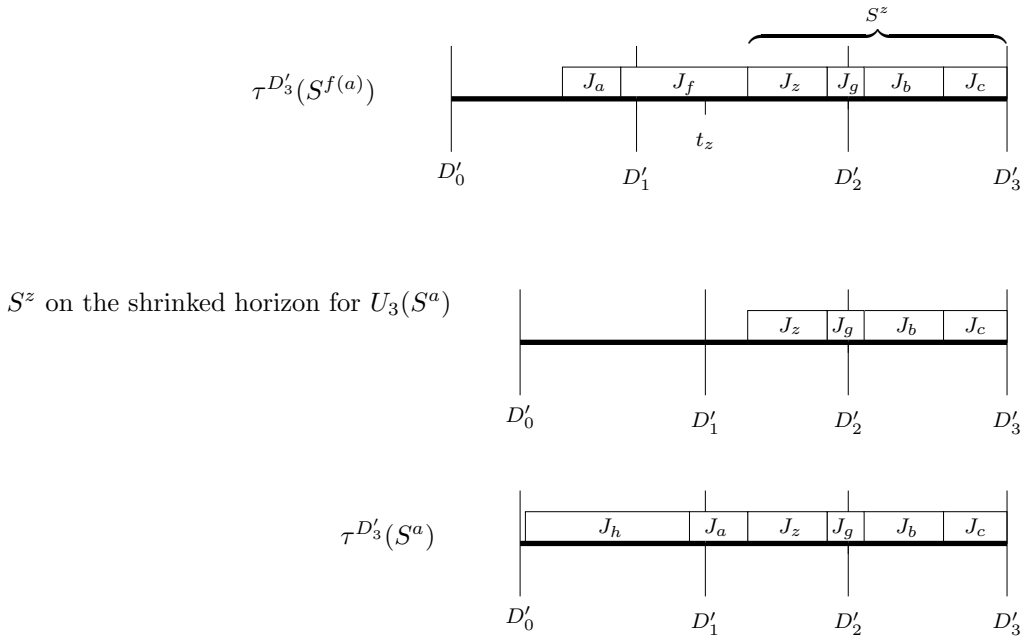


Figure 3.28: The shrunk horizons for $U_k(S^a)$ and $U_k(S^{f(a)})$, where $k = K = 3$. Between $t_z = D'_1(S^a)$ and $D'_3(S^a) = D'_3(S^{f(a)})$, the two horizons are identical.

and Bound of depth included in $[i_k, i_{k+1}[$*, it is only necessary to consider the branches* $I_k, \ldots, I_{K+1}$.

Special case of $1|r_i|V_1 + V_2$

If $K = 2$, the following structural properties apply.

Property 9. *Consider the schedule obtained by left-shifting* S^{D_2}*: if it schedules* U_1 *jobs before* D_1*, then it is an optimal schedule.*

Property 10. *If* U_1 *jobs are scheduled before* D_1 *in the current partial schedule* S^a*, then* S^a *can be completed in polynomial time with SDD-algorithm.*

Property 10 is valid because of the dominance rule for the two delivery dates problem presented in Chapter 4.

3.5 Experimentations

The algorithm is tested on randomly generated instances, as described in the next section.

3.5.1 Instances generator

The instances generator takes the following inputs: the number of jobs N, the number of delivery dates K, a parameter $A \in]0,1]$ and a parameter $R \in]0,1]$.

The processing times of the jobs are first generated, each being picked randomly from a uniform distribution $\{10,\ldots,99\}$. Then, the delivery dates are: $D_1 = \lfloor (A \times \sum_{i=1}^{N} p_i)/K \rfloor$ and $D_k = k \times D_1$, for $k = 1,\ldots,K$. The release date r_i associated with processing time p_i is chosen in one of the intervals $R_k = [D_{k-1}, D_{k-1} + R \times D_1]$, $k = 1,\ldots,K$. To avoid J_i to be trivially late (i.e. $r_i + p_i > D_K$), r_i is chosen as follows. First, determine in which intervals R_k the value r_i can be chosen. Let $k_i = \max_{k=1,\ldots,K}\{k | D_{k-1} + p_i \leq D_K\}$. An interval R_k is chosen randomly among $R_1,\ldots,R_{k_i}$ (each with a probability of $1/k_i$). Finally, r_i is picked randomly into R_k (uniform distribution), however by avoiding the possibly forbidden dates of R_{k_i}.

The parameters were chosen as follows: $K \in \{3,5,8,10,20\}$; $N = n \times K$, where $n \in \{10,20,30,50,100\}$; $A \in \{0.5,0.8,1,1.2\}$; $R \in \{0.1,0.3,0.5,0.7,1\}$.

For each of these 500 configurations, 10 instances were generated.

3.5.2 Numerical results

The tests were performed on a 2.66 GHz Intel Core2-Duo processor, 4 GB RAM, running Debian wheezy/sid. Each instance was initially allocated a maximum CPU time of 15 minutes. The experimentations were conducted on a preliminary version of the Branch and Bound algorithm in which the upper bound is computed from scratch at each node.

Results by K and n. We first present the results according to the values of K and n. Tables 3.1 to 3.5 present the percentages of instances of each of the following categories:

- instances solved at the root node (Table 3.1)
- instances solved in less than 1 second (Table 3.2)
- instances solved in less than 120 seconds (Table 3.3)
- instances solved before the time limit of 15 minutes (Table 3.4)
- instances unsolved at the end of the time limit of 15 minutes (Table 3.5)

For the last category of instances, we present the mean gap at the end of the time limit, in Table 3.6. The gap is computed as: $(upperbound - lowerbound)/upperbound$.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	85.5%	66%	39.5%	39%	14.5%
20	87%	72.5%	43.5%	38.5%	18%
30	83%	68.5%	48.5%	39%	23%
50	86.5%	67.5%	53%	46.5%	24%
100	85%	70%	54%	51%	28%

Table 3.1: Instances solved at the root node.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	100%	100%	98.5%	86.5%	42%
20	100%	99%	92.5%	87.5%	44.5%
30	100%	97%	90%	84.5%	40%
50	100%	99.5%	90%	84%	34.5%
100	100%	96%	87%	73.5%	28%

Table 3.2: Instances solved in less than 1 second.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	100%	100%	100%	92%	60%
20	100%	100%	96.5%	92.5%	60.5%
30	100%	99%	93.5%	87%	59%
50	100%	99.5%	92.5%	88%	65%
100	100%	97%	91%	91.5%	70%

Table 3.3: Instances solved in less than 120 seconds.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	100%	100%	100%	97.5%	62.5%
20	100%	100%	98%	93.5%	61%
30	100%	99.5%	96%	89%	61.5%
50	100%	100%	94%	88%	72%
100	100%	99%	91%	92%	73%

Table 3.4: Instances solved in less than 15 minutes.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	0%	0%	0%	2.5%	37.5%
20	0%	0%	2%	6.5%	39%
30	0%	0.5%	4%	11%	38.5%
50	0%	0%	6%	12%	28%
100	0%	1%	9%	8%	27%

Table 3.5: Instances unsolved at the end of the time limit of 15 minutes.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	–	–	–	1.95%	1.09%
20	–	–	0.36%	0.77%	0.57%
30	–	0.53%	0.34%	0.41%	0.38%
50	–	–	0.24%	0.27%	0.2%
100	–	0.09%	0.11%	0.11%	0.08%

Table 3.6: Mean gap of the x unsolved instances at the end of the time limit of 15 minutes, where $x =$ 200× the percentage of Table 3.5.

We see from Table 3.4 that for instances with up to 10 delivery dates and 1000 jobs, almost all the instances (at least 88% of each class) are solved in less than 15 minutes. For the instances with 20 deliver dates, still 61% of the instances are solved within the time limit. Moreover, we can notice from Tables 3.3 and 3.4 that the majority of the instances solved within the time limit are solved in less than 2 minutes.

For the last category of instances (i.e. the instances that could not be optimally solved in less than 15 minutes), we present the results when allowing 5 minutes more of computation. In Table 3.7 we present the percentage of instances that could be solved in 20 minutes, among the unsolved instances after 15 minutes. In Table 3.8 are reported

the gaps of the unsolved instances at the time limit of 20 minutes.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	–	–	–	20%	2.7%
20	–	–	25%	0%	1.3%
30	–	0%	0%	0%	0%
50	–	–	8.3%	4.2%	0%
100	–	0%	0%	0%	5.6%

Table 3.7: Instances that could be solved in 20 minutes, among the unsolved instances after 15 minutes.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	–	–	–	2.08%	1.06%
20	–	–	0.38%	0.77%	0.57%
30	–	0.53%	0.34%	0.41%	0.38%
50	–	–	0.22%	0.27%	0.2%
100	–	0.09%	0.11%	0.11%	0.08%

Table 3.8: Mean gap of the y unsolved instances at the time limit of 20 minutes, where $y = x\times$ the percentage of Table 3.7.

On Tables 3.6 and 3.8, we notice that, for a fixed number of delivery dates, the mean gaps are higher when there are less jobs. To explain this behavior, we have the following hypothesis. We show, in Chapter 4, an approximation result for the two delivery dates case which says that the initial gap is at most equal to 1. We conjecture that this result can be extended to K delivery dates: the initial gap would be at most equal to $1 + 2 + \ldots + K - 1 = K(K-1)/2$. Hence, two instances with the same number of delivery dates would have similar absolute gaps. However, the more the jobs, the less the relative gap.

We notice that 5 minutes more of computation do not have a significant impact (see Tables 3.7 and 3.8). Therefore, we attempted to close the gap for one class of instances, by allowing 1 hour of computation. We chose the class where $K = 10$ and $n = 100$, since none of the unsolved instances in 15 minutes of this class could be solved in 20 minutes; moreover there was no improvement of the mean gap, which is one of the smallest (0.11%). On the 16 instances that were allowed one hour computation, only one could be solved optimally in less than 1 hour (in 1484 seconds). The mean gap of the other 15 instances is 0.1, which is very close to the mean gap after 15 or 20 minutes.

Below, we present in Table 3.9 the mean CPU time for the instances that were not solved at the root node, and in Table 3.10 the corresponding standard deviations, computed as $\sqrt{E(T^2) - (E(T))^2}$, where $E(T)$ (resp. $E(T^2)$) is the mean CPU time (resp. the mean squared CPU time) over the concerned instances.

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	0	0	0.46	19.21	21.93
20	0	0.82	9.84	4.51	9.26
30	0	6.62	17.47	13.06	30.23
50	0.02	3.49	15.21	1.01	36.81
100	0.03	6.94	1.17	5.27	40.6

Table 3.9: Mean CPU time for the instances that were not solved at the root node (in at most 15 minutes).

$n\backslash K$	**3**	**5**	**8**	**10**	**20**
10	0	0	3.97	83.45	83.83
20	0	5.86	48.79	20.14	27.08
30	0.01	50.16	72.81	73.83	96.51
50	0.06	27.8	83.86	3.66	67.1
100	0.05	30.32	3.44	18.65	97.73

Table 3.10: Standard deviations of the CPU time for the instances that were not solved at the root node (in at most 15 minutes).

From Table 3.10 we see that the solving times are not homogeneous, as it could be seen in Tables 3.1 to 3.5. The majority of the instances is rapidly solved, however there exists a non negligible number of more difficult to solve instances.

Overall, the mean CPU time remains correct, considering that the mean CPU times of Table 3.9 only take into account the instances that were solved by exploring more than the root node.

Results by A and R. In order to identify the more difficult instance configurations, we show below the same kind of tables as above, this time according to the values of A and R.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	38%	36.4%	38.4%	32%	12.8%
0.8	46.4%	47.2%	42.8%	33.2%	14.4%
1	66.4%	65.2%	60.4%	54%	24%
1.2	97.2%	94.4%	94.4%	92.8%	74.8%

Table 3.11: Instances solved at the root node.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	71.2%	69.2%	68.8%	67.6%	68.4%
0.8	73.6%	72%	74%	72%	75.6%
1	90.4%	89.6%	87.6%	86.8%	82.8%
1.2	99.6%	100%	99.6%	100%	97.6%

Table 3.12: Instances solved in less than 1 second.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	81.6%	78%	83.6%	79.2%	82%
0.8	80.4%	79.6%	80.8%	78.4%	82.8%
1	96.4%	98%	95.6%	98.8%	95.6%
1.2	100%	100%	100%	100%	99.6%

Table 3.13: Instances solved in less than 120 seconds.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	83.6%	80%	84.4%	81.6%	84.8%
0.8	82%	81.6%	82.8%	80.4%	84.8%
1	98%	98.4%	96%	99.2%	97.2%
1.2	100%	100%	100%	100%	99.6%

Table 3.14: Instances solved in less than 15 minutes.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	16.4%	20%	15.6%	18.4%	15.2%
0.8	18%	18.4%	17.2%	19.6%	15.2%
1	2%	1.6%	4%	0.8%	2.8%
1.2	0%	0%	0%	0%	0.4%

Table 3.15: Instances unsolved at the end of the time limit of 15 minutes.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	0.39%	0.54%	0.48%	0.54%	1.24%
0.8	0.22%	0.23%	0.32%	0.34%	0.76%
1	0.07%	0.13%	0.12%	0.25%	0.29%
1.2	–	–	–	–	0.02%

Table 3.16: Mean gap of the x unsolved instances at the end of the time limit of 15 minutes, where $x = 250\times$ the percentage of Table 3.15.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	0%	0%	0%	2.2%	2.6%
0.8	4.4%	6.4%	2.3%	0%	0%
1	0%	0%	0%	0%	28.6%
1.2	–	–	–	–	0%

Table 3.17: Instances that could be solved in 20 minutes, among the unsolved instances after 15 minutes.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	0.39%	0.54%	0.48%	0.52%	1.15%
0.8	0.21%	0.23%	0.32%	0.34%	0.76%
1	0.06%	0.13%	0.1%	0.25%	0.39%
1.2	–	–	–	–	0.02%

Table 3.18: Mean gap of the y unsolved instances before the time limit of 20 minutes, where $y = x\times$ the percentage of Table 3.17.

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	15.16	16.4	10.62	16.81	13.15
0.8	16.95	11.74	22.08	13.68	11.38
1	10.88	5.22	6.2	5.02	11.88
1.2	0.51	0.06	0.32	0.13	1.2

Table 3.19: Mean CPU times for the instances that were not solved at the root node (in at most 15 minutes).

$A\backslash R$	**0.1**	**0.3**	**0.5**	**0.7**	**1**
0.5	64.16	70.07	44.86	71.48	51.97
0.8	73.23	39.18	90.92	61.07	63.73
1	39.39	20.41	22.48	20.26	66.05
1.2	0.42	0.11	0.64	0.23	5.1

Table 3.20: Standard deviations of the CPU time for the instances that were not solved at the root node (in at most 15 minutes).

We can notice that the value of R does not affect significantly the results.

However, the smallest values of A (0.5 and 0.8) correspond to instances with higher CPU times. This can be explained by noticing that when A is greater (i.e. equal to 1 or 1.2), the horizon is longer and thus more jobs can be scheduled before the last delivery date. Hence, there is less concurrence among candidate jobs to be scheduled.

If we only consider instances with small A, there are still at least 80% of each class of instances that are solved within the time limit of 15 minutes, the majority of which are solved in less than 2 minutes.

Finally, allowing 5 more minutes of computation time does not seem significant,

except for $A = 1$ and $R = 1$ (cf. Table 3.17), but this result must be considered in the light of the small number of unsolved instances within 15 minutes, for this configuration, compared to the configurations where $A = 0.5$ or $A = 0.8$ (cf. Table 3.15).

3.6 Conclusion

In this chapter, we proposed a Branch and Bound method for the single machine problem. For this purpose, we established some dominance rules, on which relies the branching scheme. Moreover, we designed some efficient bounds relying on the SDD algorithm; some pruning rules are presented to improve the Branch and Bound method. Finally, experimental results are presented on randomly generated instances with up to 20 delivery dates and 2000 jobs: 85% of all the tested instances were optimally solved in less than 2 minutes; while the mean gap over all instances unsolved after 15 minutes is less than 0.5%.

In the next chapter, we study the single machine problem with two delivery dates.

Chapter 4

Solving the single machine problem with two delivery dates

The two delivery dates problem $1|r_i|V_1 + V_2$ has been proven to be NP-hard in Chapter 2. In order to solve $1|r_i|V_1 + V_2$, we first present a specific dominance rule for this problem in Section 4.1. Then, in Section 4.2 we propose a pseudopolynomial time exact method, based on dynamic programming (which proves the weak NP-hardness of $1|r_i|V_1 + V_2$); and in Section 4.3 a polynomial time algorithm yielding a solution with an absolute performance guarantee. Finally, Section 4.4 gathers the long formal proofs of this chapter.

4.1 Structural properties

The dominance rules for $1|r_1|\sum_{k=1}^{K} V_k$ presented in Section 3.1 are also valid for $1|r_i|V_1 + V_2$, and are summarized in Property 11 below. First, recall the following notation.

NOTATION. Any (partial) schedule S of $1|r_i|\sum V_k$ can be split into $K+1$ subschedules $S_1, \ldots, S_{K+1}$; S_k being the subschedule of the jobs completing into $I_k =]D_{k-1}, D_k]$, $k = 1, \ldots, K+1$. Thus, S can be expressed as $S = S_1.S_2. \ldots .S_{K+1}$, where $S_i.S_j$ denotes the concatenation of subschedule S_i with subschedule S_j (assuming that $\mathcal{J}(S_i) \cap \mathcal{J}(S_j) = \emptyset$).

Property 11. *There exists an optimal schedule $S^* = S_1^*.S_2^*.S_3^*$ for $1|r_i|V_1 + V_2$ that is an ERD-schedule where S_2^* is a block.*

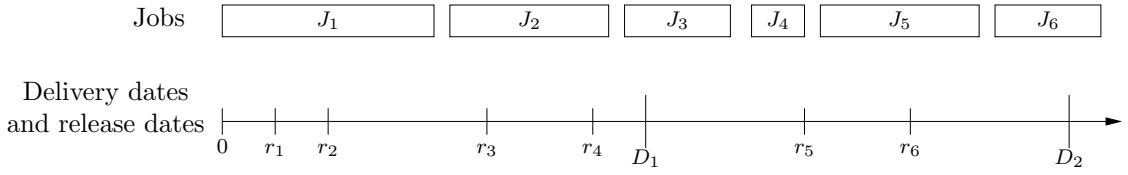
Another dominance rule, specific to $1|r_i|V_1 + V_2$, is expressed by Property 12. First, recall the following notations.

NOTATIONS. We denote by S^{D_k} the schedule obtained with $SDD(\mathcal{J}^{all}, 0, D_k)$. U_k denotes the number of jobs of S^{D_k}; hence U_k is the maximal number of jobs that can complete at or before D_k in any feasible schedule, $k = 1, \ldots, K$.

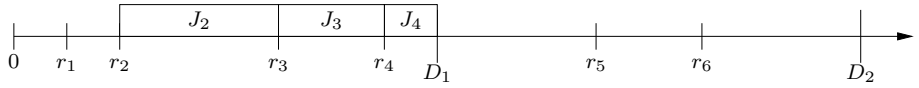
For any feasible schedule S, and, a fortiori, for an optimal schedule, we have $|\mathcal{J}(S_1)| \leq U_1$.

Property 12. *There exists an optimal schedule $S^* = S_1^*.S_2^*.S_3^*$ for $1|r_i|V_1 + V_2$ where $|\mathcal{J}(S_1^*)| = U_1$.*

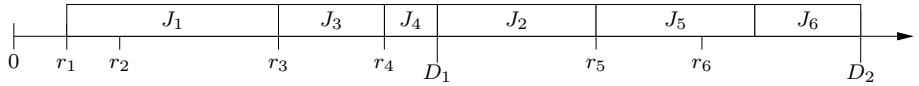
Before proving Property 12 by the means of Proposition 8, a small example is given in Figure 4.1.



(a) An instance of $1|r_i|V_1 + V_2$.



(b) $U_1 = 3$.



(c) An optimal schedule S^* where $|\mathcal{J}(S_1^*)| = U_1$.

Figure 4.1: An example to illustrate Property 12.

Proposition 8. *Given any feasible schedule S of $1|r_i|V_1 + V_2$, there exists a feasible schedule S' such that $|\mathcal{J}(S_1')| = U_1$ and $v(S') \geq v(S)$.*

The proof of Proposition 8 is based on constructive arguments. For this purpose, we introduce a constructive algorithm that, given a schedule S such that $|\mathcal{J}(S_1)| < U_1$, constructs a schedule S' such that $|\mathcal{J}(S_1')| = U_1$ and $v(S') \geq v(S)$, starting from $S' = S$. The algorithm will be called DS-algorithm (for Dominant Schedule) and will be described at page 74.

NOTATION. Let S^{D_1}-jobs denote the jobs that belong to S^{D_1}, and let $\overline{S}^{D_1}$-jobs denote the jobs of $\mathcal{J}^{all}$ that do not belong to S^{D_1}.

The key idea of the DS-algorithm is to rearrange jobs in S', by reinserting $\overline{S}^{D_1}$-jobs that are in S_1' into S_2' and S_3' and reinserting all the S^{D_1}-jobs into S_1', while maintaining feasibility and without decreasing the total payoff of S'. Left-shifting some S^{D_1}-jobs, and right-shifting some $\overline{S}^{D_1}$-jobs will thus be necessary, as on the example of Figures 4.2, 4.3, 4.4.

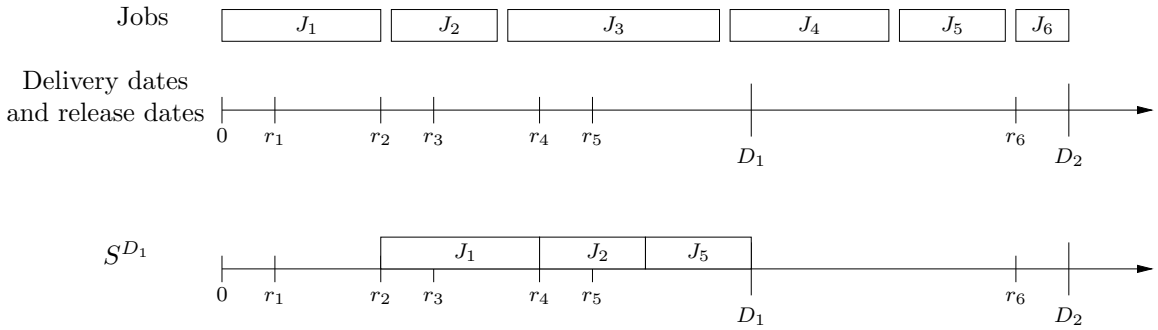
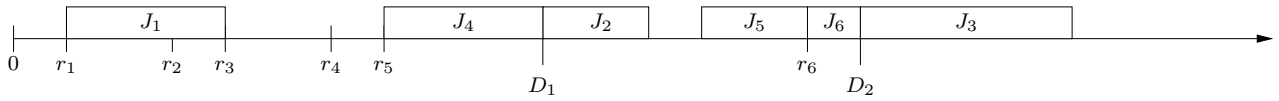
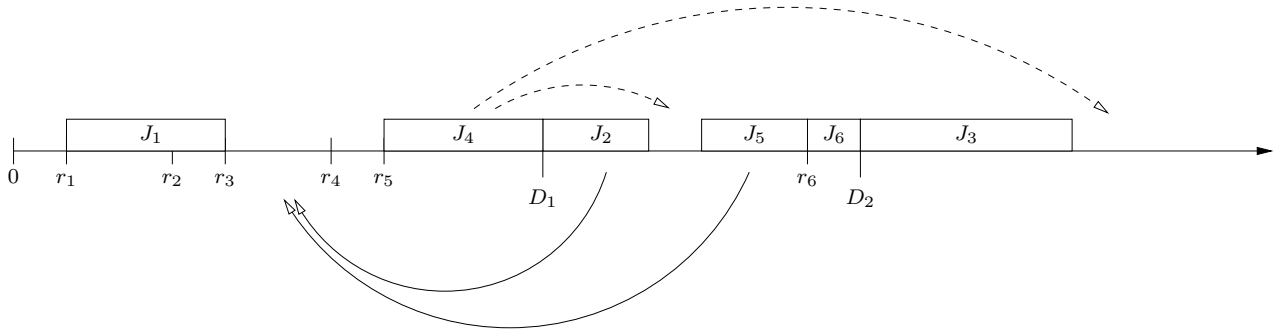


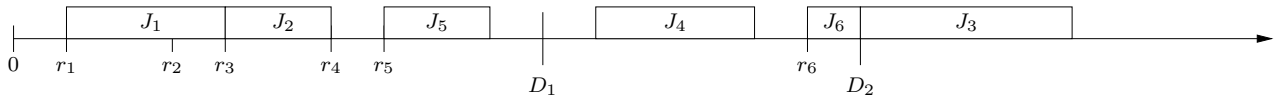
Figure 4.2: An instance of $1|r_i|V_1 + V_2$, whose $U_1 = 3$. In this example, J_1, J_2, J_5 are the S^{D_1}-jobs.



(a) A schedule S where $|\mathcal{J}(S_1)| < U_1 = 3$ and $v(S) = 7$.



(b) The two S^{D_1}-jobs J_2 and J_5 are reinserted into S_1, while the $\overline{S}^{D_1}$-job J_4 is removed from S_1.



(c) If J_4 is reinserted into S'_2, the obtained schedule S' has a payoff of 8, since the payoff variation of J_2 (resp. J_4, resp. J_5) is 1 (resp. -1, resp. 1).

(d) If J_4 is reinserted into S'_3, the obtained schedule S' has a payoff of 7, since the payoff variation of J_2 (resp. J_4, resp. J_5) is 1 (resp. -2, resp. 1).

Figure 4.3: Job exchanges to obtain a schedule where U_1 jobs complete at or before D_1, for the instance of Figure 4.2.

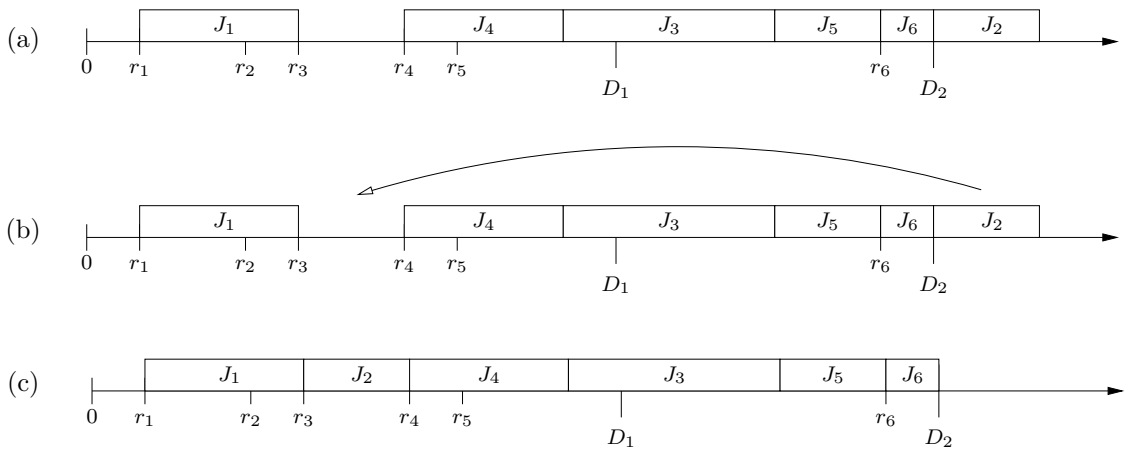


Figure 4.4: For the instance of Figure 4.2, a schedule S with less than U_1 jobs completing at or before D_1 is illustrated in (a). Suppose that J_2 is the first S^{D_1}-job that is reinserted into S_1 (b). After this reinsertion, we obtain a schedule where U_1 jobs complete at or before D_1, even if they are not all S^{D_1}-jobs (c) (in such a case, DS-algorithm stops).

Finally, note that the DS-algorithm will only be used to prove Proposition 8.

Before describing the DS-algorithm and its proof of validity, we need some preliminary lemma and definitions.

NOTATION. Let $\overline{C}^1_{max}$ be the completion time of the last $\overline{S}^{D_1}$-job in S'_1 ($\overline{C}^1_{max} = 0$ if there are no $\overline{S}^{D_1}$-jobs in S'_1).

Lemma 3. *Given a schedule S', let J_i be an S^{D_1}-job scheduled in S'_2 or S'_3. If there is no straddling job J_s starting before D_1 and completing after D_1 in S', and if the total sum of the idle times in $[\overline{C}^1_{max}, D_1]$ is at least equal to p_i, then J_i can be reinserted in S' into the interval $[\overline{C}^1_{max}, D_1]$.*

Proof. Let X be the set of S^{D_1}-jobs scheduled into $[\overline{C}^1_{max}, D_1]$ in S'. Then, $SDD(X \cup \{J_i\}, \overline{C}^1_{max}, D_1)$ schedules all the jobs of the set $X \cup \{J_i\}$ into $[\overline{C}^1_{max}, D_1]$, hence resulting in a partial schedule denoted σ^x (cf. Figure 4.5). Indeed, the interval $[\overline{C}^1_{max}, D_1]$ is wide enough to contain all the processing times of the jobs of $X \cup \{J_i\}$. However, we must be sure that σ^x is feasible (i.e. each job starts after its release date). Notice that the sequence of jobs corresponding to σ^x is a subsequence of the sequence associated to S^{D_1}. Hence, because both schedules are right-justified and complete at D_1, for any job J_j of σ^x we have $C_j(S^{D_1}) \leq C_j(\sigma^x)$. Therefore, as S^{D_1} is feasible, σ^x is also feasible. □

We define two operators that will be used in DS-algorithm to reinsert jobs inside the schedule S': a left-shift (resp. right-shift) operator $LS(J_i, x, y)$ (resp. $RS(J_i, x, y)$), which shifts J_i from S'_x to S'_y, with $x, y \in \{1, 2, 3\}$ ($y < x$ for LS and $y > x$ for RS). Let us describe these operators, and the induced payoff variations: the payoff variations

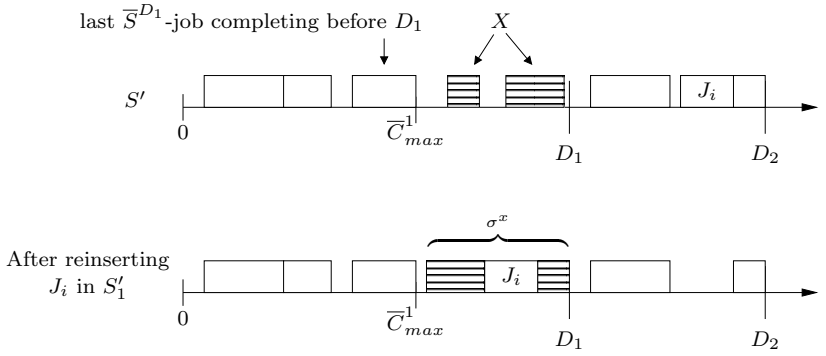
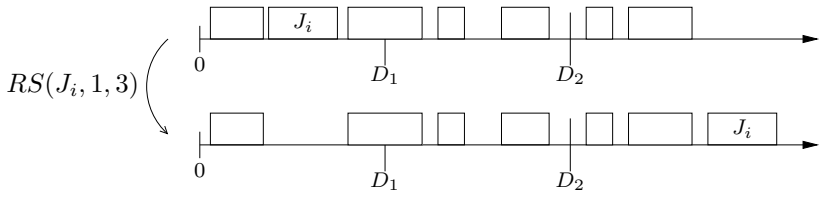


Figure 4.5: Example for Lemma 3.

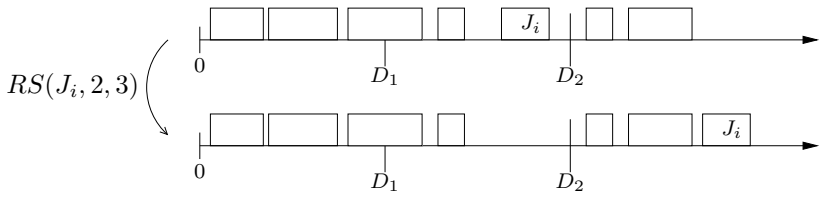
can be easily deduced by reminding that a job completing in $[0, D_1]$ is worth 2 for the payoff, a job completing in $]D_1, D_2]$ is worth 1, and a job completing after D_2 is worth 0.

1. For any job J_i we define:

   - $RS(J_i, 1, 3)$: J_i is removed from S'_1 and reinserted after the last job of S'_3, as shown below. It induces a payoff variation of -2.



   - $RS(J_i, 2, 3)$: J_i is removed from S'_2 and reinserted after the last job of S'_3, as shown below. It induces a payoff variation of -1.

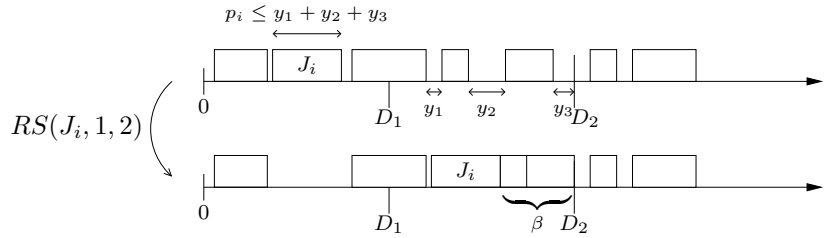


   After performing $RS(J_i, 1, 3)$ or $RS(J_i, 2, 3)$, the feasibility is maintained, as J_i is right-shifted.

2. For any job J_i such that p_i is at most equal to the total sum of the idle times in S'_2, we define:

   - $RS(J_i, 1, 2)$: J_i is removed from S'_1 and reinserted in S'_2, in the following way (see figure below). First, all the jobs of S'_2 that start at or after D_1 are right-

shifted in order to obtain a unique block β that completes at D_2. Then, J_i is inserted in S'_2 before the first job of β.



After performing $RS(J_i, 1, 2)$, the feasibility is maintained, as J_i is right-shifted and, by assumption, the idle time induced by right-shifting the jobs of S'_2 is large enough to schedule J_i. $RS(J_i, 1, 2)$ induces a payoff variation of -1.

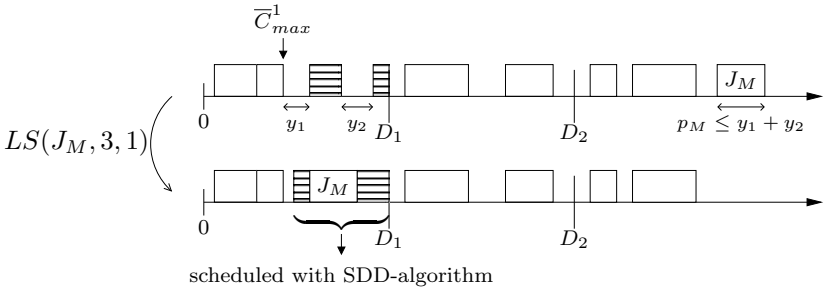
3. For any S^{D_1}-job J_M, with processing time p_M, such that:

   (a) there is no straddling job in S' starting before D_1 and completing after D_1, and

   (b) the total sum of the idle times between $\overline{C}^1_{max}$ and D_1 in S' is at least p_M,

   we define:

   - $LS(J_M, 3, 1)$: J_M is removed from S'_3 and reinserted in S'_1 into the interval $[\overline{C}^1_{max}, D_1]$, as shown below.



   In the example of the above figure, the striped jobs are the S^{D_1}-jobs scheduled after $\overline{C}^1_{max}$ in S'_1. J_M is reinserted in S'_1 by rescheduling all the striped jobs and J_M between $\overline{C}^1_{max}$ and D_1 with the SDD-algorithm. After $LS(J_M, 3, 1)$ has been applied, the feasibility is maintained by Lemma 3, i.e. release dates are satisfied. $LS(J_M, 3, 1)$ induces a payoff variation of 2.

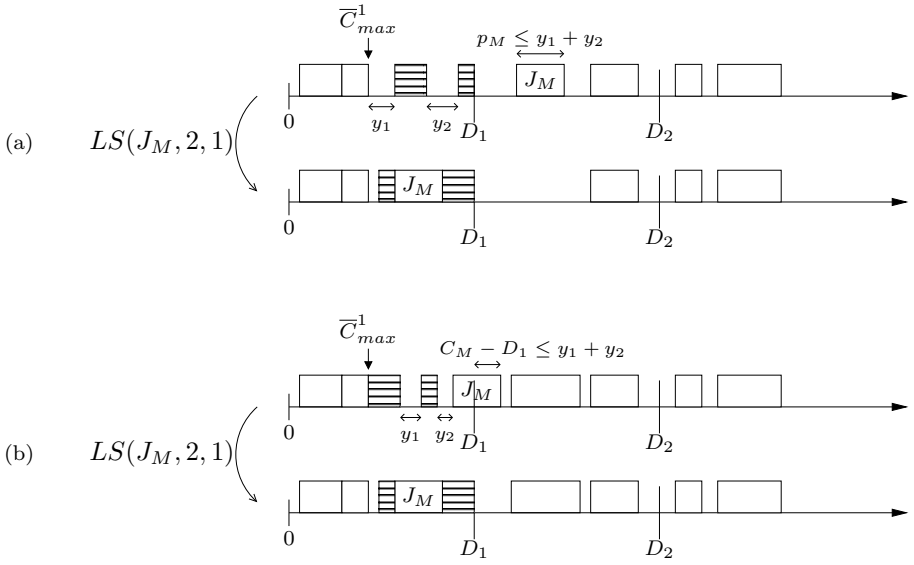
4. For any S^{D_1}-job J_M, with processing time p_M, such that:

   (a) there is no straddling job in S' starting before D_1 and completing after D_1, and the total sum of the idle times between $\overline{C}^1_{max}$ and D_1 in S' is at least p_M, or

(b) J_M is a straddling job starting before D_1 and completing after D_1, and the total sum of the idle times between $\overline{C}^1_{max}$ and b_M is at least $C_M - D_1$,

we define:

- $LS(J_M, 2, 1)$: J_M is removed from S'_2 and reinserted in S'_1 into the interval $[\overline{C}^1_{max}, D_1]$ (see the two figures below, where the striped jobs are the S^{D_1}-jobs scheduled after $\overline{C}^1_{max}$ in S'_1.).



J_M is reinserted in S'_1 by rescheduling all the striped jobs and J_M between $\overline{C}^1_{max}$ and D_1 with the SDD-algorithm. After $LS(J_M, 2, 1)$ has been applied, the feasibility is maintained, by Lemma 3, i.e. release dates are satisfied. Indeed, in the case in which J_M is a straddling job, once J_M is removed from S'_2 there is no straddling job anymore, and therefore Lemma 3 applies. $LS(J_M, 2, 1)$ induces a payoff variation of 1 in both cases.

DS-algorithm is a recursive function, depicted by pseudocode Algorithm 8 (p. 74), that takes as parameters the initial schedule $S' = S'_1.S'_2.S'_3$ and q, the current step of the algorithm.

NOTATION. We denote by $\overline{n}_1$ the number of $\overline{S}^{D_1}$-jobs in S'_1 and by n_2 (resp. n_3) the number of S^{D_1}-jobs in S'_2 (resp. S'_3).

We have $|\mathcal{J}(S'_1)| = U_1 + \overline{n}_1 - (n_3 + n_2)$, thus the terminal condition is achieved when $\overline{n}_1 = n_2 + n_3$. Initially, $\overline{n}_1 < n_2 + n_3$.

We describe next the structure of the algorithm.

There are two main cases: the initial case $q = 0$ (lines 2-14) and the general case $q > 0$ (lines 15-40). For each of these two cases, we have two subcases. Therefore, we

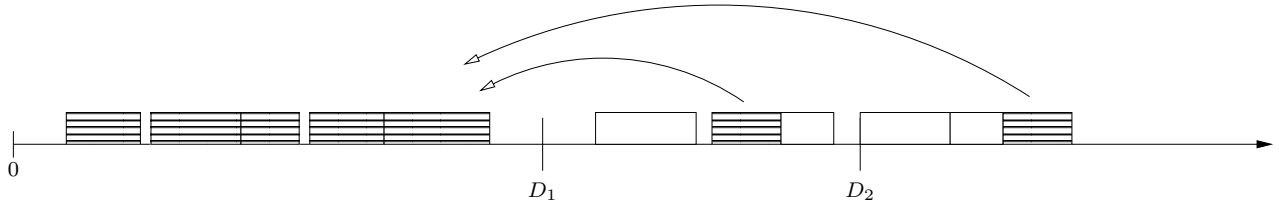
```
   Algo8: q, S'
1  if n̄_1 = n_2 + n_3 then return S'
2  if q = 0 then
3      if (n̄_1 = 0 and n_2 ≥ 0) or (n̄_1 > 0 and n_2 ∈ {0,1}) then
4          for each S̄^{D_1}-job J_i in S'_1 do RS(J_i, 1, 3)
5          if ∃ a straddling S̄^{D_1}-job J_s then RS(J_s, 2, 3)
6          for each S^{D_1}-job J_M in S'_2 do LS(J_M, 2, 1)
7          for each S^{D_1}-job J_M in S'_3 do LS(J_M, 3, 1)
8      else
9          if (∃ a straddling job J_s) and (∃ an S^{D_1}-job J_i in S'_2 s.t. (r_i ≤ b_s and p_i ≤ D_1 − b_s) or
           (r_i > b_s)) then
10             RS(J_s, 2, 3)
11             LS(J_i, 2, 1)
12         if n̄_1 < n_2 + n_3 then
13             if n_2 ∈ {0,1} then return Algo8(0, S')
14             else return Algo8(1, S')
15 else
16     if ∀ E_{q+1}, p(G_q) < p(E_{q+1}) then
17         if n̄_1 > q and n_2 > q + 1 then return Algo8(q + 1, S')
18         else
19             if n_2 = q + 1 and there is a straddling S^{D_1}-job J_s then flag ← true
20             else flag ← false
21             for each S^{D_1}-job J_M in S'_2 do RS(J_M, 2, 3)
22             if ∃ a straddling job J_s then RS(J_s, 2, 3)
23             for each job J_j in G_{q−1} do RS(J_j, 1, 2)
24             J_u ← the unique job of G_q\G_{q−1}
25             if flag then RS(J_u, 1, 3)
26             else RS(J_u, 1, 2)
27             for each S̄^{D_1}-job J_i in S'_1 do RS(J_i, 1, 3)
28             for each S^{D_1}-job J_M in S'_3 do LS(J_M, 3, 1)
29     else
30         if ∃ a straddling job J_s such that J_s ∉ E_{q+1} then
31             J_f ← the first job of E_{q+1}
32             exchange J_f and J_s
33         for each job J_i in E_{q+1} do RS(J_i, 2, 3)
34         for each job J_j in G_{q−1} do RS(J_j, 1, 2)
35         J_u ← the unique job of G_q\G_{q−1}
36         RS(J_u, 1, 3)
37         for each job J_i in E_{q+1} do LS(J_i, 3, 1)
38         if n̄_1 < n_2 + n_3 then
39             if n̄_1 > 0 and n_2 ≥ 2 then return Algo8(1, S')
40             else return Algo8(0, S')
41 return S'
```

Algorithm 8. DS-algorithm.

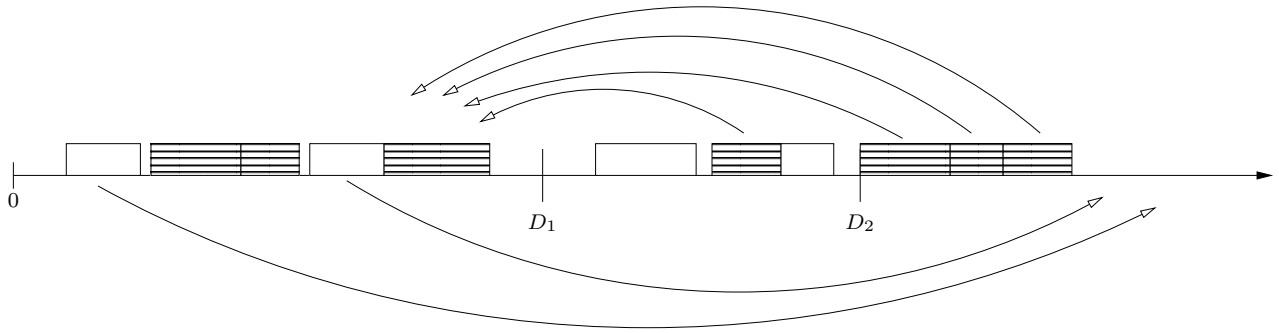
consider Algorithm 8 as composed of four parts: Part 1 includes lines 3-7, Part 2 lines 8-14, Part 3 lines 16-28 and Part 4 lines 29-40.

In the initial case, if there are no $\overline{S}^{D_1}$-jobs in S'_1 or if there is at most one S^{D_1}-job in

S'_2 (lines 3-7), then it is easy, without decreasing the payoff, to reinsert all the S^{D_1}-jobs in S'_1 (and possibly reinsert some $\overline{S}^{D_1}$-jobs in S'_3) (see Figure 4.6). Otherwise, if there exists a straddling job starting before D_1 and completing after D_1 and it is possible to exchange it with another job in order to obtain a schedule without straddling job (and without decreasing the payoff), this exchange is performed at lines 9-11 (see Figure 4.7).



(a) If $\overline{n}_1 = 0$, it is sufficient to reinsert all the S^{D_1}-jobs of S'_2 and S'_3 to S'_1: the payoff increases (by $n_2 + 2n_3$).



(b) If $\overline{n}_1 > 0$ and $n_2 \leq 1$, all the S^{D_1}-jobs of S'_2 and S'_3 are reinserted into S'_1, while all the $\overline{S}^{D_1}$-jobs in S'_1 and possibly a straddling $\overline{S}^{D_1}$-job in S'_2 are reinserted into S'_3. The payoff variation is equal to $n_2 + 2n_3 - 2\overline{n}_1$. In this example, since $n_2 = 1$, and since $\overline{n}_1 < n_2 + n_3$, the payoff increases.

Figure 4.6: The initial case of Algorithm 8: the striped jobs are the S^{D_1}-jobs.

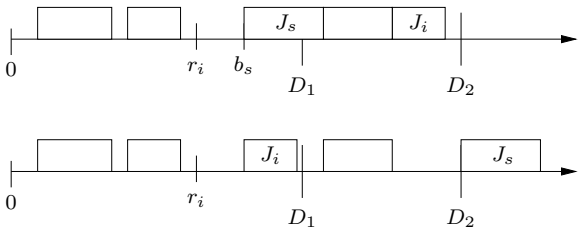
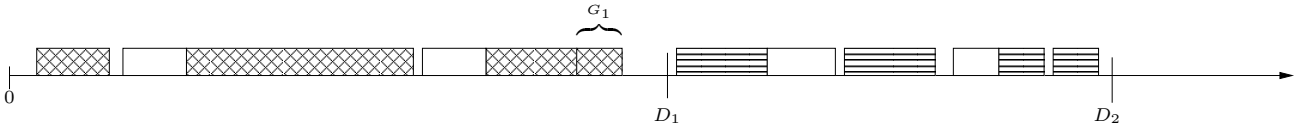


Figure 4.7: "Removal" of the straddling job.

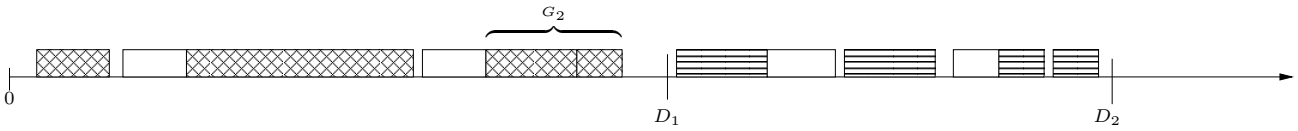
The general case is composed of parts 3 and 4. Let us give the idea of its inductive structure (refer to Figure 4.8). At the beginning of each step $q > 0$, we consider G_q: the set of the last q $\overline{S}^{D_1}$-jobs scheduled in S'_1. We also denote by E_{q+1} a set of $q + 1$ S^{D_1}-jobs in S'_2. Then, if the following condition H_q is satisfied, we execute Part 3:

$$\forall\ E_{q+1},\ p(G_q) < p(E_{q+1}) \qquad (H_q)$$

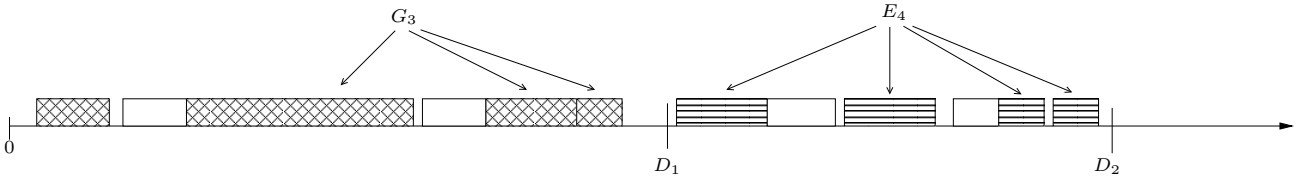
Otherwise, we execute Part 4.



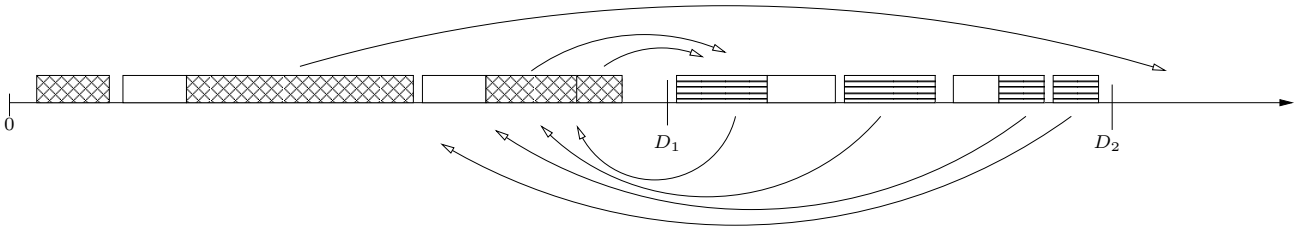
(a) For any pair E_2 of S^{D_1}-jobs in S'_2, H_1 is satisfied: $p(G_1) < p_(E_2)$.



(b) For any set E_3 of three S^{D_1}-jobs in S'_2, H_2 is satisfied: $p(G_2) < p_(E_3)$.



(c) There exists a set E_4 of four S^{D_1}-jobs in S'_2, such that $p(G_3) \geq p(E_4)$.



(d) The following reinsertions can be done: first, the job of $G_3 \backslash G_2$ is reinserted into S'_3. Then, the jobs of G_2 and E_4 can be exchanged, since $p(G_2) < p(E_3)$ by H_2 and since $p(E_4) \leq p(G_3)$. The total payoff variation is 0, since the payoff variation of the job of $G_3 \backslash G_2$ (resp. the jobs of G_2, resp. the jobs of E_4) is -2 (resp. -2, resp. 4). The number of S^{D_1}-jobs in S'_1 increases by 1.

Figure 4.8: The general case of Algorithm 8: the crossed jobs are $\overline{S}^{D_1}$-jobs, while the striped jobs are S^{D_1}-jobs.

Of these two cases, the only one in which the instruction "go to step $q+1$" (represented by the call of $Algo8(q+1, S')$) can be performed is Part 3 (at line 17). Thus, (H_q) is an inductive hypothesis. Indeed, if we are at a given step $q_u > 1$, then we executed Part 3 in all the previous steps q such that $1 \leq q < q_u$ (cf. Figures 4.8a, 4.8b and 4.8c). Hence, at any step $q_u > 1$, (H_q) is observed for all $1 \leq q < q_u$.

When performing Part 4, there exists an E_{q+1} that satisfies condition $p(G_q) \geq p(E_{q+1})$ (cf. Figures 4.8c and 4.8d). Therefore, by removing the jobs of G_q from S'_1, the jobs of E_{q+1} can be reinserted in S'_1, and consequently the number of jobs in S'_1 increases. This condition can be reached several times before the terminal condition is satisfied, and each time $|\mathcal{J}(S'_1)|$ increases.

Finally, correctness of Algorithm 8 is established by Proposition 9, which is proven in Section 4.4.1.

Proposition 9. *Algorithm 8 with parameters* $(0, S)$ *yields a feasible schedule* S' *whose payoff is greater than or equal to* $v(S)$*, and that schedules* U_1 *jobs completing no later than* D_1*.*

4.2 Exact method

In this section, we describe a dynamic programming algorithm for $1|r_i|V_1 + V_2$, and then show the experimental results of this algorithm on randomly generated instances.

4.2.1 Dynamic programming algorithm

The key point of the dynamic programming algorithm is to construct ERD-schedules where S_2 is a block (cf. Property 11, p. 11).

Any schedule S will be described by a 4-tuple $(C_{max}(S_1), B_2(S), C_{max}(S_2), v(S))$. Of course this 4-tuple does not provide a precise description of the schedule, like completion times of the jobs; however it includes enough information for the purpose of the algorithm, which is to return an optimal payoff. In this section, we first handle schedules, to explain the idea of the algorithm, then we define some formal functions that handle 4-tuples.

The jobs are reindexed in ERD order: $J_1 \prec_{ERD} \cdots \prec_{ERD} J_N$. There are N steps in the dynamic programming algorithm. At each step $j = 1, \ldots, N$, we construct a set $\mathcal{S}_j$ of dominant ERD-schedules where S_2 is a block, starting out from $\mathcal{S}_{j-1}$, the set of schedules built at the previous step. To build $\mathcal{S}_j$, we modify the schedules of $\mathcal{S}_{j-1}$ by reinserting job J_j either before D_1 or between D_1 and D_2, or keeping J_j after D_2. Indeed, the initial set of schedules $\mathcal{S}_0$ contains only one schedule, in which all the jobs are scheduled after D_2, in ERD order. The unique schedule of $\mathcal{S}_0$ is represented by the following 4-tuple: $(0, D_2, 0, 0)$. In this 4-tuple, $C_{max}(S_1) = 0$ because S_1 is empty, therefore it can be represented as starting at time 0 and completing at time 0. Since S_2 is empty too, it should be represented as starting at D_1 and finishing at D_1; however, we set its starting time at D_2 ($B(S_2) = D_2$) and its completion time at 0 ($C_{max}(S_2) = 0$), in order to avoid considering additional subcases. This assumption does not affect the correctness of the algorithm.

Suppose now we are at step j, $j = 1, \ldots, N$. Let $S = S_1.S_2.S_3$ be a schedule of $\mathcal{S}_{j-1}$. Then, $J_j \in \mathcal{J}(S_3)$. Indeed, as a job J_j is reinserted only at step j, job J_j is scheduled after D_2 in all the schedules of $\mathcal{S}_{j-1}$. Thus, starting out from S, we can build some new schedules (at most three) by reinserting job J_j in three different ways: in S_1, in S_2, or in S_3.

When J_j is reinserted into S_1, resp. S_2, it is always scheduled at the end of S_1, resp. S_2, in order to obtain an ERD-schedule (see Figure 4.9).

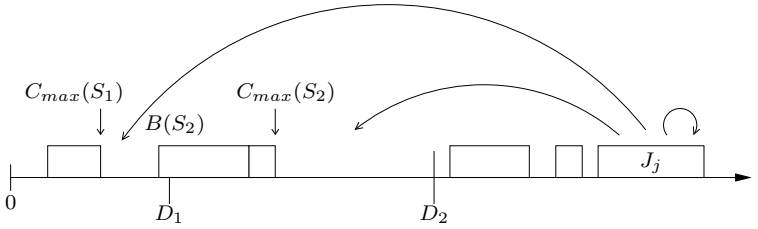


Figure 4.9: The three ways job J_j can be reinserted into S.

Moreover, if we attempt to reinsert J_j into S_1:

- if J_j can be scheduled after $C_{max}(S_1)$ while completing before both D_1 and the starting time of the first job of S_2 (i.e. if $max(r_j, C_{max}(S_1))+p_j \leq min(B(S_2), D_1)$), then J_j is scheduled as soon as possible after $C_{max}(S_1)$ and we have $C_j = max(r_j, C_{max}(S_1)) + p_j$.
- if J_j can be scheduled after $C_{max}(S_1)$ while completing before D_1 (i.e. $max(r_j, C_{max}(S_1))+ p_j \leq D_1$) but cannot complete before the starting time of the first job of S_2 (i.e. if $max(r_j, C_{max}(S_1)) + p_j > B(S_2)$), and if S_2 can be right-shifted of the necessary number of time units to allow J_j to be reinserted in S_1 while keeping the completion times of all the jobs of S_2 in I_2 (i.e. $C_{max}(S_2) + max(C_{max}(S_1), r_j) + p_j - B(S_2) \leq D_2$), then this right-shift is performed and J_j is reinserted in S_1 (see Figure 4.10).
- otherwise, J_j is not reinserted in S_1.

In the first two cases, the payoff of the new schedule is $v(S) + 2$.

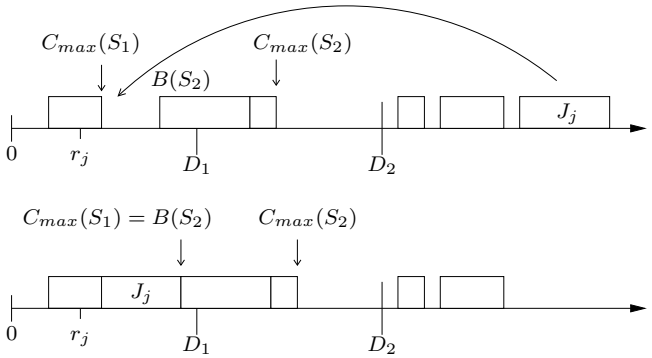


Figure 4.10: An example in which J_j is reinserted into S_1: S_2 is right-shifted by $max(C_{max}(S_1), r_j) + p_j - B(S_2)$ time units to allow J_j to be scheduled.

Consider now the case in which we attempt to reinsert J_j into S_2.

NOTATION. Let b_j^{min} be the earliest starting time allowing job J_j to complete after D_1: $b_j^{min} = \max(r_j, D_1 - p_j + 1)$ (see Figure 4.11).

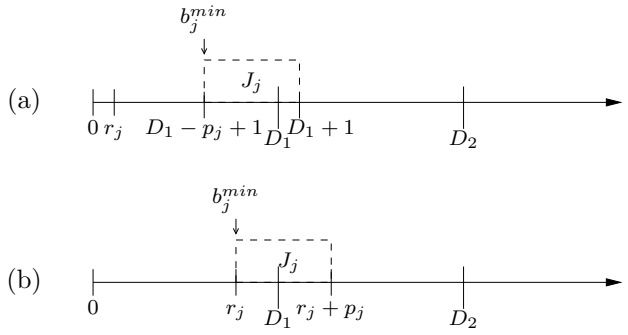


Figure 4.11: Value of b_j^{min} if: (a) $r_j < D_1 - p_j + 1$, or (b) $r_j \geq D_1 - p_j + 1$.

- if S_2 is empty, and if $\max(b_j^{min}, C_{max}(S_1)) + p_j \leq D_2$, then J_j is scheduled as soon as possible while completing into $]D_1, D_2]$; thus $C_j = \max(b_j^{min}, C_{max}(S_1)) + p_j$.
- if S_2 is not empty, and if J_j can be scheduled after $C_{max}(S_2)$ while completing into $]D_1, D_2]$ (i.e. $max(r_j, C_{max}(S_2)) + p_j \leq D_2$), then J_j is scheduled as soon as possible after $C_{max}(S_2)$; thus $C_j = max(r_j, C_{max}(S_2)) + p_j$. If $C_{max}(S_2) < r_j$, then all the jobs of S_2, except for J_j, are right-shifted in order to constitute a unique block with J_j, while maintaining feasibility (see Figure 4.12).
- otherwise, J_j is not reinserted in S_2.

In the first two cases, the payoff of the new schedule is $v(S) + 1$.

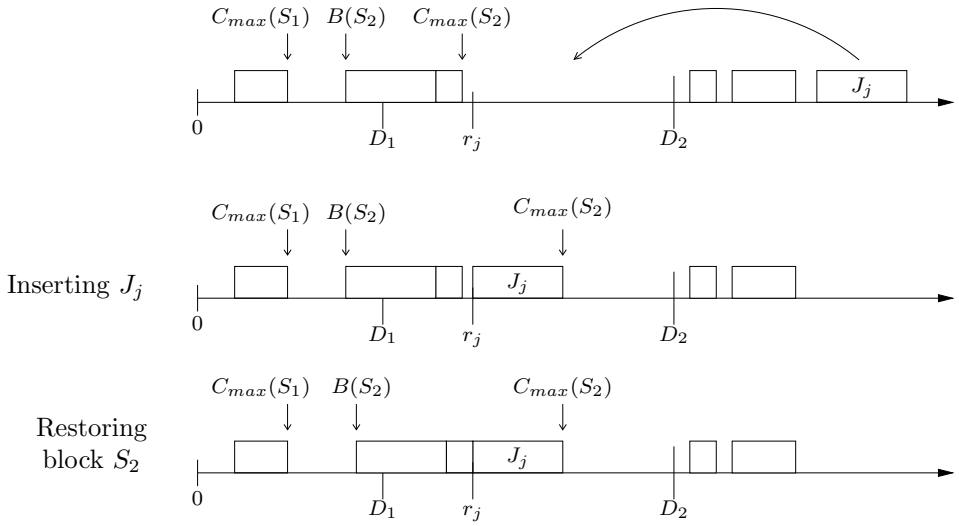


Figure 4.12: An example in which J_j is reinserted into S_2: the previous block in S_2 is right-shifted by $r_j - C_{max}(S_2)$ time units to avoid idle times in S_2.

An example of the execution of the algorithm on a small instance is given in Figure 4.13.

More formally, let $\mathcal{Q}_j$ be the set of 4-tuples corresponding to the schedules of $\mathcal{S}_j$, $j = 0, \ldots, N$. Then, at step j, we can define three functions g_1, g_2, g_3 on $\mathcal{Q}_{j-1}$. Given a 4-tuple $(C_{max}(S_1), B_2(S), C_{max}(S_2), v(S))$ of $\mathcal{Q}_{j-1}$, corresponding to a solution S of $\mathcal{S}_{j-1}$, function g_1 (resp. g_2, g_3) returns the 4-tuple associated to a solution obtained by reinserting job J_j in S_1 (resp. in S_2, in S_3).

Before giving the formal definition of these three functions, we need the following notations. For a given $j \in \{1, \ldots, N\}$ and a given schedule $S \in \mathcal{S}_{j-1}$:

- α is the earliest possible completion time of J_j if it is reinserted in S_1: $\alpha = \max(r_j, C_{max}(S_1)) + p_j$,
- β_1 is the earliest possible completion time of J_j if it is reinserted in S_2 when S_2 is empty: $\beta_1 = \max(b_j^{min}, C_{max}(S_1)) + p_j$,
- β_2 is the earliest possible completion time of J_j if it is reinserted in S_2 when S_2 is not empty: $\beta_2 = \max(r_j, C_{max}(S_2)) + p_j$.

NOTATION. For shortness, we denote $v(S)$ by v, when no ambiguity is possible.

The formal definitions of functions g_1, g_2, g_3 follow.

$$g_1(C_{max}(S_1), B(S_2), C_{max}(S_2), v, j) = \begin{cases} (\alpha, B(S_2), C_{max}(S_2), v+2) & \text{if } \alpha \leq \min(B(S_2), D_1) \\ (\alpha, \alpha, C_{max}(S_2) + \alpha - B(S_2), v+2) & \text{if } B(S_2) < \alpha \leq D_1 \text{ and} \\ & C_{max}(S_2) + \alpha - B(S_2) \leq D_2 \\ (C_{max}(S_1), B(S_2), C_{max}(S_2), v) & \text{otherwise} \end{cases}$$

$$g_2(C_{max}(S_1), B(S_2), C_{max}(S_2), v, j) = \begin{cases} (C_{max}(S_1), \beta_1 - p_j, \beta_1, v+1) & \text{if } B(S_2) = D_2 \text{ and } \beta_1 \leq D_2 \\ (C_{max}(S_1), B(S_2) + \beta_2 - p_j - C_{max}(S_2), \beta_2, v+1) & \text{if } B(S_2) < D_2 \text{ and } \beta_2 \leq D_2 \\ (C_{max}(S_1), B(S_2), C_{max}(S_2), v) & \text{otherwise} \end{cases}$$

$$g_3(C_{max}(S_1), B(S_2), C_{max}(S_2), v, j) = (C_{max}(S_1), B(S_2), C_{max}(S_2), v)$$

Two 4-tuples $(C_{max}(S_1), B(S_2), C_{max}(S_2), v(S))$ and $(C_{max}(S_1'), B_2(S_1'), C_{max}(S_2'), v(S'))$ are said to be similar iff $C_{max}(S_1) = C_{max}(S_1')$ and $B_2(S) = B_2(S')$ and $C_{max}(S_2) = C_{max}(S_2')$. Clearly, at step j, some subsets of similar 4-tuples can be generated with functions g_1, g_2, g_3. In this case, we keep in $\mathcal{Q}_j$ only one of the similar 4-tuples, that has the maximal value of v.

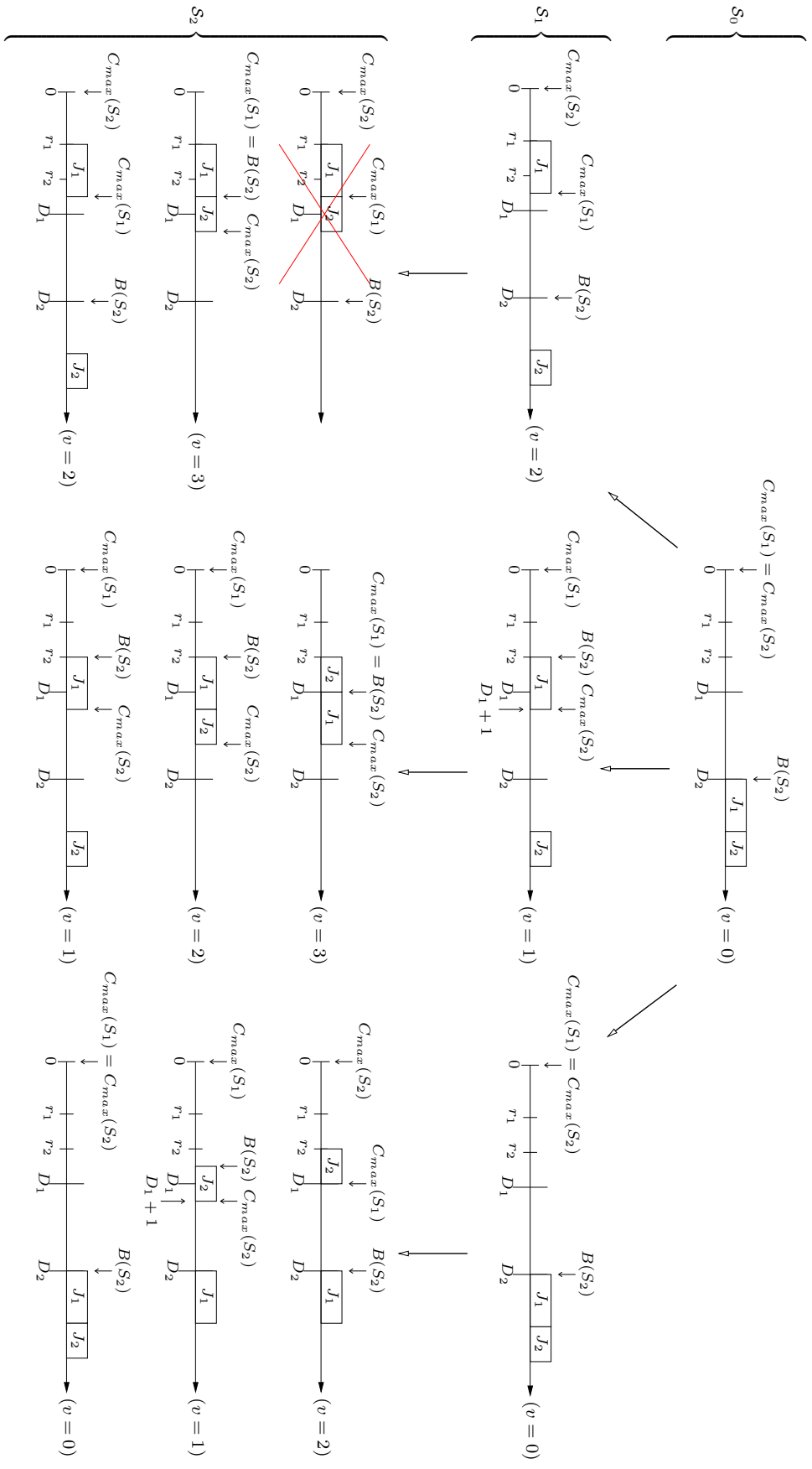


Figure 4.13: An execution of the dynamic programming algorithm.

The algorithm ends after N steps. Once $\mathcal{Q}_N$ is built (without similar 4-tuples), the maximal value of v among those of all the 4-tuples of $\mathcal{Q}_N$ is an optimal payoff.

The complexity of the algorithm is given by the sizes of the sets $\mathcal{Q}_1, \ldots, \mathcal{Q}_N$. Thus, it is given by the number of steps (N) and the maximal number of non-similar 4-tuples built at each step. The number of non-similar 4-tuples generated at a step is bounded by $O(D_1(D_2)^2)$, since the number of possible values of $C_{max}(S_1)$ (resp. $B(S_2)$, resp. $C_{max}(S_2)$) is bounded by $O(D_1)$ (resp. $O(D_2)$, resp. $O(D_2)$). Moreover, sorting the jobs in ERD order at the beginning of the algorithm induces a complexity of $O(NlogN)$.

Finally, the total complexity is $O(NlogN + ND_1(D_2)^2)$. Since this complexity is pseudopolynomial, and since we showed in Chapter 2 that $1|r_i|V_1 + V_2$ is NP-hard, we deduce that $1|r_i|V_1 + V_2$ is weakly NP-hard.

Remark: Notice that the dynamic programming algorithm can be extended to any number K of delivery dates, by adapting the number and the definitions of the g_i functions. In this case, we have a variable $C_{max}(S_1)$ for the first delivery date, and two variables $B(S_k)$ and $C_{max}(S_k)$ for each of the other delivery dates ($k = 2, \ldots, K$). Hence, the time complexity of the dynamic programming algorithm for K delivery dates is $O(NlogN + ND_1(D_K)^{2(K-1)})$. When K is fixed, this complexity remains pseudopolynomial. Thus we deduce that $1|r_i, \text{fixed } K|\sum_{k=1}^{K} V_k$ is weakly NP-hard.

The pseudocode of the dynamic programming algorithm, its proof of correctness, and the formal proof of its complexity are given in Section 4.4.2.

For the sake of clarity, we provided an algorithm that returns an optimal payoff, and only handles sets $\mathcal{Q}_j$. However, it is easy to modify it in order to get an optimal schedule (by also handling sets $\mathcal{S}_j$).

In order to speed up the execution of the algorithm, we introduce a pruning rule for the dynamic programming algorithm. Recall that the jobs are reindexed in ERD order.

Pruning rule. *Let J_e be the last job such that its release date is strictly less than D_1: $e = \max_{i=1,\ldots,N}\{i|r_i < D_1\}$. At the end of step e of the dynamic programming algorithm, prune all the schedules that do not have exactly U_1 jobs completing in the interval $[0, D_1]$.*

The pruning rule is based on Property 12, which states that the set of schedules where U_1 jobs complete before D_1 is dominant.

4.2.2 Experimentations

The dynamic programming algorithm was tested on randomly generated instances. Instances are generated as seen in Section 3.5.1 for the Branch and Bound. We recall that

the generator takes the following inputs:

- the number of jobs N,
- a parameter $A \in]0,1]$, representing the ratio $D_2/\sum_{i=1}^{N} p_i$,
- a parameter $R \in]0,1]$ (release dates are picked randomly in the intervals $[0, R \times D_1]$ and $[D_1, D_1 + R \times D_1]$).

For each combination of $N \in \{30, 40, 50, 60, 70, 80\}$, $A \in \{0.8, 1, 1.2\}$ and $R \in \{0.1, 0.3, 0.5\}$, 5 instances were generated. Some experiments were performed to analyze the efficiency of the algorithm in terms of CPU time. The tests were performed in C++ on a 3.33 GHz Intel Core2-Duo processor, 8 GB RAM, running Debian wheezy/sid. Each run was limited to a single processor.

Since the results did not show that the CPU times are strongly related to the values of a and R, we only present the results in relation to the number of jobs N, in Table 4.1.

N	Number of solved instances	CPU time
30	45	9.3
40	45	56.7
50	45	274.8
60	44	668.4
70	28	797.8
80	9	1186.6

Table 4.1: For each value of N, the number of instances (out of 45 instances) solved within a time limit of 30 minutes CPU, and the mean CPU time (in seconds) for the solved instances.

Table 4.1 shows that it becomes hard to efficiently solve instances with more than 70 jobs within reasonable CPU time. Indeed, for the 70-jobs instances, seventeen instances require more than 30 minutes to be solved.

Some further tests were performed to study the influence of D_1 and D_2 on the efficiency of the algorithm. We fixed $N = 40$ and $R = 0.3$. Moreover, we considered the following parameters, where B represents ratio D_1/D_2:

- $A \in \{0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1, 1.2, 1.5\}$
- $B \in \{0.1, 0.3, 0.4, 0.6, 0.8, 0.9\}$

For each pair of values of A and B, 70 instances are generated, as above, except for the delivery dates, which are generated from parameters A and B. The average CPU times are provided in Table 4.2.

$A\backslash B$	**0.1**	**0.3**	**0.4**	**0.6**	**0.8**	**0.9**
0.3	2	9.4	11.8	12.3	5.2	2.8
0.4	5.7	21.5	28.2	24.1	13.1	4.4
0.5	8.1	32.2	43.8	45.1	16.5	5.5
0.7	16.5	57.8	70	58.9	28.9	8.6
0.8	22.5	78.1	72.3	55.4	30.8	10.9
0.9	26.9	81.4	83.4	43.5	30.4	13.5
1	31.3	84.5	68.4	46.2	32.1	11
1.2	38.1	91.8	58.9	45.7	28.3	12.4
1.5	60.1	71.7	54.8	31.1	24.5	11.2

Table 4.2: For each couple (A, B), mean CPU time on 70 instances (in seconds).

We can notice that when D_1 is very small or very large, compared to D_2 ($B = 0.1$ or $B = 0.9$), the CPU time is usually shorter than for other values of B. This can be explained by noticing that, in these two cases, the instances are very similar to the single delivery date problem $1|r_i|V$. For the same reason, the instances with $B = 0.8$ are also quite easy. Conversely, the most difficult instances are those generated with $B = 0.3$ or $B = 0.4$.

As for parameter A, we notice that when it increases, the CPU time increases up to some point, then it decreases. Since an increase of A implies an increase of D_2, the initial increase of CPU times can be explained by observing that the complexity of the algorithm is $O(NlogN + ND_1(D_2)^2)$. However, when D_2 is very large, almost all the jobs can be scheduled before D_2, and therefore it is easier to obtain an optimal solution. This can explain the observed decrease in the CPU times.

4.3 Polynomial algorithm with performance guarantee

In this section, we show how to obtain a near-optimal feasible solution for $1|r_i|V_1 + V_2$ using a polynomial time algorithm. We prove that the difference between the payoff of the obtained feasible schedule and the optimal payoff is at most 1.

A feasible schedule $S^l = S_1^l.S_2^l.S_3^l$ is obtained with Algorithm 9, which is equivalent to Algorithm 2 (p. 42) when $K = 2$. Recall that, for any feasible (partial) schedule S, we denote by left-shift(S) the left-shifted (partial) schedule that schedules the jobs of S in the same order (cf. the definition of a left-shifted schedule, p. 26).

Notice that, before the left-shifting of S_1^l, $S_1^l = S^{D_1}$, since both schedules are obtained as the output of $SDD(\mathcal{J}^{all}, 0, D_1)$.

NOTATION. Let $S_{1,2}^l = S_1^l.S_2^l$ be the partial schedule equal to S^l between 0 and D_2.

Input: $\mathcal{J}^{all}$, D_1, D_2
Output: S^l, l_bound
1 $\mathcal{J} \leftarrow \mathcal{J}^{all}$
2 $S_1^l \leftarrow SDD(\mathcal{J}, 0, D_1)$
3 $S_1^l \leftarrow$ left-shift(S_1^l)
4 $\mathcal{J} \leftarrow \mathcal{J} \backslash \mathcal{J}(S_1^l)$
5 $S_2^l \leftarrow SDD(\mathcal{J}, C_{max}(S_1^l), D_2)$
6 $\mathcal{J} \leftarrow \mathcal{J} \backslash \mathcal{J}(S_2^l)$
7 $S_3^l \leftarrow$ a schedule that schedules the jobs of $\mathcal{J}$ in ERD order after D_2
8 **return** $S_1^l.S_2^l.S_3^l$, $2|\mathcal{J}(S_1^l)| + |\mathcal{J}(S_2^l)|$

Algorithm 9: Construction of a feasible schedule for $1|r_i|V_1 + V_2$.

Notice that $v(S_{1,2}^l) = v(S^l)$, because the jobs completing after D_2 do not provide any payoff.

To prove the approximation guarantee, we will compare $v(S_{1,2}^l)$ to the upper bound u_bound obtained with Algorithm 10, which yields an upper bound on the optimal payoff of $1|r_i|V_1 + V_2$, since it is equivalent to Algorithm 3 (p. 43), when $K = 2$.

Input: $\mathcal{J}^{all}$, D_1, D_2
Output: u_bound
1 $S^{D_1} \leftarrow SDD(\mathcal{J}^{all}, 0, D_1)$
2 $U_1 \leftarrow |\mathcal{J}(S^{D_1})|$
3 $S^{D_2} \leftarrow SDD(\mathcal{J}^{all}, 0, D_2)$
4 $U_2 \leftarrow |\mathcal{J}(S^{D_2})|$
5 $u\_bound \leftarrow U_1 + U_2$
6 **return** u_bound

Algorithm 10: Computation of an upper bound on the optimal payoff of $1|r_i|V_1 + V_2$.

The main result of this section is stated in Theorem 4.

Theorem 4. *Algorithm 9 yields a feasible schedule S^l for $1|r_i|V_1 + V_2$ such that $v(S^l) \geq v(S^*) - 1$, where S^* is an optimal schedule for $1|r_i|V_1 + V_2$.*

In order to prove the result of Theorem 4, we will prove that $u\_bound - v(S^l) = u\_bound - v(S_{1,2}^l) \leq 1$.

We have: $v(S_{1,2}^l) = 2|\mathcal{J}(S_1^l)| + |\mathcal{J}(S_2^l)| = |\mathcal{J}(S_1^l)| + |\mathcal{J}(S_{1,2}^l)|$. Since $\mathcal{J}(S_1^l) = \mathcal{J}(S^{D_1})$, $v(S_{1,2}^l) = |\mathcal{J}(S^{D_1})| + |\mathcal{J}(S_{1,2}^l)| = U_1 + |\mathcal{J}(S_{1,2}^l)|$.

Moreover, $u\_bound = U_1 + U_2$. Hence, $u\_bound - v(S_{1,2}^l) = U_2 - |\mathcal{J}(S_{1,2}^l)|$.

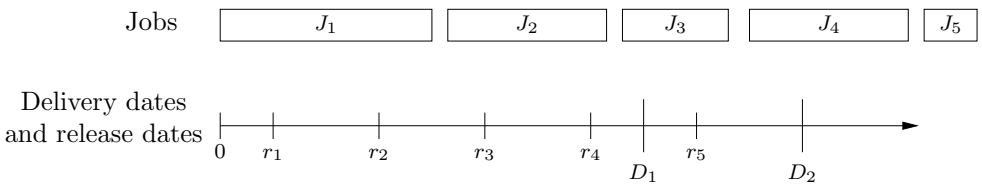
Therefore, we will show that $U_2 - |\mathcal{J}(S_{1,2}^l)| \leq 1$. Notice that we are comparing the number of jobs of two feasible schedules: U_2 is the number of jobs of $S^{D_2} = SDD(\mathcal{J}^{all}, 0, D_2)$, and $|\mathcal{J}(S_{1,2}^l)|$ is the number of jobs of the feasible schedule $S_{1,2}^l$.

Hence, we will prove Theorem 5, that directly implies the result of Theorem 4.

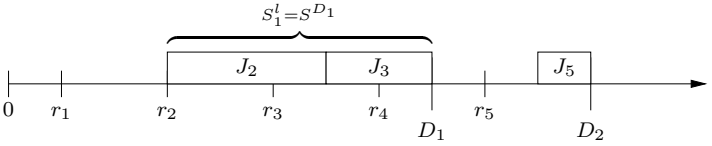
Theorem 5. $U_2 - |\mathcal{J}(S^l_{1,2})| \leq 1$.

The formal proof of Theorem 5 can be seen in Section 4.4.3. Here, we attempt to give its rationale.

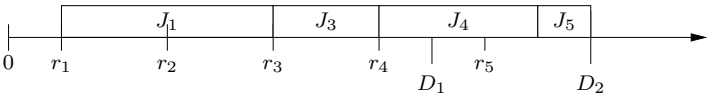
By Property 12 (p. 68), there exists an optimal schedule S^* of $1|r_i|V_1 + V_2$ such that $|\mathcal{J}(S^*_1)| = U_1 = |\mathcal{J}(S^l_1)|$. Therefore, $v(S^*) - v(S^l_{1,2}) \leq 1$ is equivalent to $|\mathcal{J}(S^*_2)| - |\mathcal{J}(S^l_2)| \leq 1$.



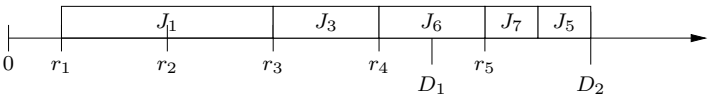
(a) An instance of $1|r_i|V_1 + V_2$.



(b) A feasible (partial) schedule $S^l_{1,2}$ obtained with Algorithm 9.



(c) An optimal schedule S^* where $|\mathcal{J}(S^*_1)| = U_1$.



(d) If two jobs (J_6 and J_7) could be scheduled in the "extra place", then at least one of the two jobs (J_7) would have been scheduled in $S^l_{1,2}$.

Figure 4.14: An example where $|\mathcal{J}(S^*_2)| - |\mathcal{J}(S^l_2)| = 1$.

Let us consider the example of Figure 4.14, where $|\mathcal{J}(S^*_2)| - |\mathcal{J}(S^l_2)| = 1$. As can be seen on this example, $S^l_{1,2}$ can be suboptimal because while constructing it we consider the intervals $[0, D_1]$ and $[C_{max}(S^l_1), D_2]$ separately, without a global vision on the total horizon. Indeed, in the example, $p(\mathcal{J}(S^*_1)) > p(\mathcal{J}(S^l_1))$ but the release dates configuration induces $C_{max}(S^*_1) < C_{max}(S^l_1)$, which allows the addition of J_4 as a straddling job. However, we cannot have $|\mathcal{J}(S^*_2)| - |\mathcal{J}(S^l_2)| \geq 2$. Let us attempt to give an intuitive reason for this. As seen on the example, the presence of an additional job is due to the

fact that $C_{max}(S_1^*) < C_{max}(S_1^l)$. Therefore, these $C_{max}(S_1^l) - C_{max}(S_1^*)$ time units allow a straddling job to be scheduled between $C_{max}(S_1^*)$ and $B(S_2^*)$. This extra place can only be used to schedule one straddling job. On the example, if there were two more jobs J_6 and J_7 that could be scheduled instead of J_4 in S^*, then one of them would have been scheduled in $S_{1,2}^l$ (see Figure 4.14d). Indeed, if another job, besides the straddling job, could be scheduled, it would not be straddling and it would be scheduled in S_2^* (since the maximal number U_1 is already attained in S_1^*).

4.4 Formal proofs

4.4.1 Correctness of the Dominant Schedule algorithm

Proposition 9 *Algorithm 8 with parameters* $(0, S)$ *yields a feasible schedule* S' *whose payoff is greater than or equal to* $v(S)$*, and that schedules* U_1 *jobs completing no later than* D_1*.*

Proof. We need to prove the following assertions:

1. The algorithm always terminates.
2. The returned schedule is feasible.
3. In the returned schedule, exactly U_1 jobs complete no later than D_1.
4. The returned schedule has a payoff greater than or equal to the payoff of the initial schedule.

Assertion 1: The algorithm always terminates. When $Algo8(0, S')$ is first called, Part 1 (lines 3-7) and/or Part 2 (lines 8-14) are executed. If we are in Part 1, the algorithm terminates, as there are no other calls to function DS. In Part 2, the algorithm terminates, unless there is a call to $Algo8(0, S')$ (line 13) or $Algo8(1, S')$ (line 14). $Algo8(0, S')$ will cause the execution of Part 1 (since $n_2 \in \{0, 1\}$), which will terminate. Therefore we only need to prove that $Algo8(1, S')$ terminates, and more generally that $Algo8(q, S')$ terminates, for $q > 0$.

Let us first consider the evolution of $\overline{n}_1, n_2, n_3$ on lines 19-28 and 30-37. At line 21, n_3 increases by n_2, and n_2 becomes equal to 0; at line 23, $\overline{n}_1$ decreases by $q - 1$, and on lines 26-27 it decreases by 1; at line 27, $\overline{n}_1$ becomes equal to 0; at line 28, n_3 becomes equal to 0. As for lines 30-37, we have: at line 33, n_3 increases by $q+1$, and n_2 decreases by $q + 1$; at line 34, $\overline{n}_1$ decreases by $q - 1$; at line 36, $\overline{n}_1$ decreases by 1; at line 37, n_3 decreases by $q + 1$.

Let us consider all the calls of DS to show that they cannot be executed indefinitely. When $Algo8(0, S')$ is called at line 40, the algorithm stops, since $n_2 \in \{0, 1\}$. Hence

we only need to examine the calls of $Algo8(1, S')$ on lines 14 and 39, and the call of $Algo8(q+1, S')$ on line 17. The instruction of line 14 is executed at most once, that is at the first call of $Algo8(0, S')$, since the following calls of $Algo8(0, S')$ are executed only if $n_2 \in \{0, 1\}$. As for the instruction of line 39, it can be executed only while $\overline{n}_1 > 0$ and $n_2 \geq 2$, which is a limited number of times, since $\overline{n}_1$ and n_2 strictly decrease on lines 19-28 and 30-37. Finally, the instruction $Algo8(q+1, S')$ of line 17 cannot be executed indefinitely, since at some point we will have $q = \overline{n}_1$ or $q + 1 = n_2$.

Assertion 2: The returned schedule is feasible. We need to show that each performed operation (LS or RS) is feasible.

Let us examine each part of the pseudocode. We will refer to the different cases considered for the definitions of the RS and LS operations in Section 4.1. Moreover, by straddling job (J_s) we always mean a job starting before D_1 and completing after D_1.

Part 1.

Line 4: the $RS(J_i, 1, 3)$ operations are feasible (cf. Case 1). After executing this line, there are no more $\overline{S}^{D_1}$-jobs in S'_1, and therefore $\overline{C}^1_{max} = 0$.

Line 5: $RS(J_s, 2, 3)$ is feasible (cf. Case 1). After executing this line, if there is a straddling job J_s, J_s is an S^{D_1}-job.

Line 6: if there is a straddling job J_s, the operation $LS(J_s, 2, 1)$ is first performed (since J_s is an S^{D_1}-job), and then we perform $LS(J_M, 2, 1)$ for each of the other S^{D_1}-jobs J_M in S'_2. Since $\overline{C}^1_{max} = 0$ and J_s is an S^{D_1}-job, the total sum of the idle times between $\overline{C}^1_{max}$ and b_s is at least $C_s - D_1$. Therefore, $LS(J_s, 2, 1)$ is feasible (cf. Case 4.(b)). Hence, when $LS(J_M, 2, 1)$ is performed for each of the non-straddling S^{D_1}-jobs J_M in S'_2, there is no straddling job. Moreover, since $\overline{C}^1_{max} = 0$, and all of these jobs are S^{D_1}-jobs, their total processing time is at most the total sum of the idle times in $[\overline{C}^1_{max}, D_1]$. Therefore, the $LS(J_M, 2, 1)$ operations are feasible (cf. Case 4.(a)).

Line 7: $LS(J_M, 3, 1)$ is performed for every S^{D_1}-job in S'_3. Since the reinserted jobs are all S^{D_1}-jobs, and since there is no straddling job, and $\overline{C}^1_{max} = 0$, the total sum of the idle times in $[\overline{C}^1_{max}, D_1]$ is at least equal to the total processing time of those jobs. Therefore, the $LS(J_M, 3, 1)$ operations are feasible (cf. Case 3).

Part 2.

Line 10: $RS(J_s, 2, 3)$ is feasible (cf. Case 1). After this operation there is no straddling job. Moreover, $\overline{C}^1_{max}$ is unchanged and $\overline{C}^1_{max} \leq b_s$.

Line 11: since J_i is an S^{D_1}-job, $r_i > b_s$ implies $p_i \leq D_1 - b_s$. Therefore, if the condition of line 9 is true, $p_i \leq D_1 - b_s$. Hence, since $\overline{C}^1_{max} \leq b_s$, the total amount of idle times in $[\overline{C}^1_{max}, D_1]$ is greater than p_i. Therefore, since there is no straddling job, $LS(J_i, 2, 1)$ is feasible (cf. Case 4.(a)).

Part 3.

Lines 19-20: notice that if *flag* is true, there are exactly $q+1$ S^{D_1}-jobs in S'_2 and thus there exists a unique E_{q+1}, which contains all the S^{D_1}-jobs in S'_2, including J_s. (If flag is false, $n_2 \geq q+1$).

Line 21: the $RS(J_M, 2, 3)$ operations are feasible (cf. Case 1). If *flag* is true, there is no straddling job after these operations. If *flag* is false, it is possible that there is a straddling job J_s after these operations: in this case, J_s is an $\overline{S}^{D_1}$-job.

Line 22: $RS(J_s, 2, 3)$ is feasible (cf. Case 1). This operation is never performed if *flag* is true. After line 22 there is no straddling job.

Lines 23-26: there are two cases, depending on the value of *flag*.

- If *flag* is false, we perform for all the jobs J_j of G_q: $RS(J_j, 1, 2)$ (lines 23 and 26). Since *flag* is false, at least $q+1$ non-straddling jobs were moved to S'_3 at line 21. Let us consider E_{q+1} a set of $q+1$ non-straddling jobs that were moved to S'_3 at line 21. So, after lines 21 and 22, the total sum of the idle times in S'_2 is at least equal to $p(E_{q+1})$. Moreover, by induction hypothesis, $p(G_q) < p(E_{q+1})$. Therefore, the total sum of the idle times in S'_2 is at least equal to $p(G_q)$. We deduce (cf. Case 2) that $RS(J_j, 1, 2)$ is feasible for all the jobs J_j of G_q.

- If *flag* is true, we perform $RS(J_j, 1, 2)$ (line 23) only for the jobs of G_{q-1}, while for the unique job J_u of $G_q \backslash G_{q-1}$ we perform $RS(J_u, 1, 3)$ (line 25). Since *flag* is true, exactly $q+1$ jobs were moved to S'_3 at line 21, one of them being J_s. Therefore, exactly q non-straddling jobs were moved to S'_3 at line 21. Let E_q be the set of those jobs. Moreover, the instruction of line 22 is not performed, since $RS(J_s, 2, 3)$ is already performed at line 21. So, after lines 21 and 22, the total sum of the idle times in S'_2 is at least equal to $p(E_q)$. By induction hypothesis, $p(G_{q-1}) < p(E_q)$. Therefore, the total sum of the idle times in S'_2 is at least equal to $p(G_{q-1})$. We deduce (cf. Case 2) that $RS(J_j, 1, 2)$ is feasible for all the jobs J_j of G_{q-1}. Finally, the operation $RS(J_u, 1, 3)$ at line 25 is feasible (cf. Case 1).

Line 27: the $RS(J_i, 1, 3)$ operations are feasible (cf. Case 1). After these operations, there are no more $\overline{S}^{D_1}$-jobs in S'_1, therefore $\overline{C}^1_{max} = 0$.

Line 28: $LS(J_M, 3, 1)$ is performed for every S^{D_1}-job in S'_3. Since the reinserted jobs are all S^{D_1}-jobs, and since there is no straddling job, and $\overline{C}^1_{max} = 0$, the total sum of the idle times in $[\overline{C}^1_{max}, D_1]$ is at least equal to the total processing time of those jobs. Therefore, the $LS(J_M, 3, 1)$ operations are feasible (cf. Case 3).

Part 4

Recall that the following condition is observed: $\exists\ E_{q+1},\ p(G_q) \geq p(E_{q+1})$.

Lines 30-32: if the condition of line 30 is true, we have that for all $J_i \in E_{q+1}, r_i \leq b_s$ and $p_i > D_1 - b_s$, because of the instructions of lines 9-11. Indeed, none of the operations performed in the algorithm transforms a schedule without straddling job into a schedule

with a straddling job, as can be seen in the definitions of the left-shift and right-shift operators. Therefore, any job J_i of E_{q+1} can be rescheduled in order to start at time b_s, by right-shifting J_s. After this exchange, J_i is the actual straddling job.

Line 33: the $RS(J_i, 2, 3)$ operations are feasible (cf. Case 1). After executing these operations there is no straddling job.

Line 34: since at least q non-straddling jobs were moved to S'_3 at line 33, the total sum of the idle times in S'_2 is at least equal to $p(E_q)$, with E_q a set of q non-straddling jobs moved to S'_3 at line 33. By induction hypothesis, $p(G_{q-1}) < p(E_q)$. Therefore, performing $RS(J_j, 1, 2)$ on every job J_j of G_{q-1} is feasible (cf. Case 2).

Line 36: $RS(J_u, 1, 3)$ is feasible (cf. Case 1).

Line 37: at lines 34-36, all the jobs of G_q were removed from S'_1. Therefore, the total amount of idle times in $[\overline{C}^1_{max}, D_1]$ is at least $p(G_q)$. Since $p(G_q) \geq p(E_{q+1})$, the total amount of idle times in $[\overline{C}^1_{max}, D_1]$ is at least $p(E_{q+1})$. For this reason, and since there is no straddling job, we deduce that performing $LS(J_i, 3, 1)$ on every job J_i of E_{q+1} is feasible (cf. Case 3).

Assertion 3: In the returned schedule, exactly U_1 jobs complete no later than D_1. Since the terminal condition $\overline{n}_1 = n_2 + n_3$ implies that exactly U_1 jobs complete no later than D_1, it is sufficient to prove that whenever the algorithm terminates, the terminal condition is true.

In Part 1, at line 4, $\overline{n}_1$ becomes equal to 0, at line 6, n_2 becomes equal to 0, and at line 7, n_3 becomes equal to 0. Therefore, the terminal condition is true.

In Part 2, the algorithm stops only if the condition $\overline{n}_1 < n_2 + n_3$ (line 12) is not satisfied, which implies that the terminal condition is true, since we cannot have $\overline{n}_1 > n_2 + n_3$, otherwise there would be a contradiction on U_1 being the maximal number of jobs that can complete no later than D_1.

If Part 3 terminates (lines 19-28), we have that $\overline{n}_1 = n_2 = n_3 = 0$, as shown in the proof of Assertion 1. Therefore the terminal condition is true.

Finally, Part 4 terminates only if the condition $\overline{n}_1 < n_2 + n_3$ (line 38) is not satisfied, which, as said above, implies that the terminal condition is true.

Assertion 4: The returned schedule has a payoff greater than or equal to the payoff of the initial schedule. In order to prove this assertion, we show that the execution of any aforementioned part does not decrease the payoff. Let $\overline{n}_1^b$ (resp. n_2^b, n_3^b) be the value of $\overline{n}_1$ (resp. n_2, n_3) at the beginning of the sequence of instructions of a given part. Let us consider each part.

- lines 4-7: at line 4, $\overline{n}_1^b$ operations $RS(J_i, 1, 3)$ are performed, inducing a payoff variation of $-2\overline{n}_1^b$; if $RS(J_s, 2, 3)$ is performed at line 5, it induces a payoff variation of -1; the n_2^b $LS(J_M, 2, 1)$ operations of line 6 induce a payoff variation of n_2^b; and

the n_3^b $LS(J_M, 3, 1)$ operations of line 7 induce a payoff variation of $2n_3^b$. Thus, the total payoff variation is at least $n_2^b + 2n_3^b - 2\overline{n}_1^b - 1$. If $\overline{n}_1^b = 0$, the payoff is at least $n_2^b + 2n_3^b - 1 \geq 0$, since $0 = \overline{n}_1^b < n_2^b + n_3^b$. Otherwise (i.e. $\overline{n}_1^b > 0$), if $n_2^b = 0$, then $\overline{n}_1^b < n_2^b + n_3^b = n_3^b$ and therefore $n_2^b + 2n_3^b - 2\overline{n}_1^b - 1 \geq 0$; else (i.e. $n_2^b = 1$), then $\overline{n}_1^b < n_3^b + 1$. Thus, $2n_3^b \geq 2\overline{n}_1^b$. Therefore, $n_2^b + 2n_3^b - 2\overline{n}_1^b - 1 \geq 0$.

- lines 10-11: the $RS(J_s, 2, 3)$ operation at line 10 induces a payoff variation of -1, while the $LS(J_i, 2, 1)$ operation at line 11 induces a payoff variation of 1. Therefore, the total payoff variation is 0.

- lines 19-28: at line 21, we perform n_2^b times the operation $RS(J_M, 2, 3)$ (payoff variation: $-n_2^b$).

  If *flag* is true, there is no straddling job after the operation of line 21. In this case, at line 23 we perform $q-1$ times the operation $RS(J_j, 1, 2)$ (payoff variation: $-q+1$) and at line 25 the operation $RS(J_u, 1, 3)$ (payoff variation: -2). Otherwise, if *flag* is false, the operation $RS(J_s, 2, 3)$ at line 22 can possibly be performed (payoff variation: -1); and q $RS(J_j, 1, 2)$ operations (payoff variation: $-q$) are performed at lines 23 and 26.

  Finally, at line 27 we perform $\overline{n}_1^b - q$ times the operation $RS(J_i, 1, 3)$ (payoff variation: $-2(\overline{n}_1^b - q)$); and at line 28 we perform $(n_2^b + n_3^b)$ times $LS(J_M, 3, 1)$ (payoff variation: $2(n_2^b + n_3^b)$).

  Therefore, the total payoff variation is at least $2n_3^b + n_2^b + q - 2\overline{n}_1^b - 1$. If $\overline{n}_1^b = q$, the payoff variation is at least $2n_3^b + n_2^b - \overline{n}_1^b - 1 \geq 0$, since $\overline{n}_1^b < n_2^b + n_3^b$. If $n_2^b = q + 1$, the payoff variation is at least $2(n_3^b + n_2^b - \overline{n}_1^b - 1) \geq 0$, since $\overline{n}_1^b < n_2^b + n_3^b$.

- lines 30-37: the exchange performed in lines 30-32 does not change the payoff; at line 33 we perform $q+1$ times the operation $RS(J_i, 2, 3)$ (payoff variation: $-q-1$); at line 34 we perform $(q-1)$ times the operation $RS(J_j, 1, 2)$ (payoff variation: $-q+1$); at line 36 we perform $RS(J_u, 1, 3)$ (payoff variation: -2); and at line 37, $(q+1)$ times the operation $LS(J_i, 3, 1)$ (payoff variation: $2(q+1)$). Therefore, the total payoff variation is 0. □

4.4.2 The dynamic programming algorithm

The pseudocode of the dynamic programming algorithm is given in Algorithm 11. In Algorithm 11, a 4-tuple $(C_{max}(S_1), B(S_2), C_{max}(S_2), v)$ is represented as a pair of a 3-tuple and a payoff: $(\langle C_{max}(S_1), B(S_2), C_{max}(S_2)\rangle, v)$, in order to easily handle similar 4-tuples. Hence, two pairs (e, v) and (e', v') are said to be similar iff $e = e'$ (note that their payoffs may differ, i.e. v is not necessarily equal to v').

Moreover, we defined in Section 4.2.1 variables β_1 and β_2:

- β_1 is the earliest possible completion time of J_j if it is reinserted in S_2 and S_2 is empty: $\beta_1 = \max(b_j^{min}, C_{max}(S_1)) + p_j$; notice that $\beta_1 = \max(b_j^{min}, C_{max}(S_1)) + p_j = \max(b_j^{min}, C_{max}(S_1), C_{max}(S_2)) + p_j$ since $C_{max}(S_2) = 0$.

- β_2 is the earliest possible completion time of J_j if it is reinserted in S_2 and S_2 is not empty: $\beta_2 = \max(r_j, C_{max}(S_2)) + p_j$; notice that $\beta_2 = \max(r_j, C_{max}(S_2)) + p_j = \max(b_j^{min}, C_{max}(S_2)) + p_j$, since $C_{max}(S_2) > D_1$; and $\max(b_j^{min}, C_{max}(S_2)) + p_j = \max(b_j^{min}, C_{max}(S_2), C_{max}(S_1)) + p_j$, since $C_{max}(S_1) < C_{max}(S_2)$.

Thus, in Algorithm 11, we use variable $\beta = max(b_j^{min}, C_{max}(S_1), C_{max}(S_2)) + p_j$ instead of β_1 and β_2.

Input: p_i, r_i (jobs sorted by ERD order), D_1, D_2
Output: The payoff of an optimal schedule

$\mathcal{Q}_0 \leftarrow \{(\langle 0, D_2, 0\rangle, 0)\}$
for $j = 1$ **to** N **do**
  `// Reinserting task` J_j `into` S_3
  $\mathcal{Q}_j \leftarrow \mathcal{Q}_{j-1}$
  `// Reinserting task` J_j `into` S_1
  foreach $(\langle C_{max}(S_1), B(S_2), C_{max}(S_2)\rangle, v) \in \mathcal{Q}_{j-1}$ **do**
    $\alpha \leftarrow \max(r_j, C_{max}(S_1)) + p_j$
    $e \leftarrow \begin{cases} \langle \alpha, B(S_2), C_{max}(S_2)\rangle & \text{if } \alpha \le B(S_2) \\ \langle \alpha, \alpha, C_{max}(S_2) + \alpha - B(S_2)\rangle & \text{otherwise} \end{cases}$
    if $C_{max}(S_1)(e) \le D_1$ *and* $C_{max}(S_2)(e) \le D_2$ **then**
      $\mathcal{Q}_j \leftarrow \begin{cases} \mathcal{Q}_j \cup \{(e, v+2)\} \backslash \{(e, v')\} & \text{if } \exists (e, v') \in \mathcal{Q}_j \text{ such that } v' < v+2 \\ \mathcal{Q}_j & \text{if } \exists (e, v') \in \mathcal{Q}_j \text{ such that } v' \ge v+2 \\ \mathcal{Q}_j \cup \{(e, v+2)\} & \text{otherwise} \end{cases}$
  `// Reinserting task` J_j `into` S_2
  foreach $(\langle C_{max}(S_1), B(S_2), C_{max}(S_2)\rangle, v) \in \mathcal{Q}_{j-1}$ **do**
    $\beta \leftarrow \max(b_j^{min}, C_{max}(S_1), C_{max}(S_2)) + p_j$
    $e \leftarrow \begin{cases} \langle C_{max}(S_1), \beta - p_j, \beta\rangle & \text{if } B(S_2) = D_2 \\ \langle C_{max}(S_1), B(S_2) + \beta - p_j - C_{max}(S_2), \beta\rangle & \text{otherwise} \end{cases}$
    if $C_{max}(S_1)(e) \le D_1$ *and* $C_{max}(S_2)(e) \le D_2$ **then**
      $\mathcal{Q}_j \leftarrow \begin{cases} \mathcal{Q}_j \cup \{(e, v+1)\} \backslash \{(e, v')\} & \text{if } \exists (e, v') \in \mathcal{Q}_j \text{ such that } v' < v+1 \\ \mathcal{Q}_j & \text{if } \exists (e, v') \in \mathcal{Q}_j \text{ such that } v' \ge v+1 \\ \mathcal{Q}_j \cup \{(e, v+1)\} & \text{otherwise} \end{cases}$
return $\max\{v : (e, v) \in \mathcal{Q}_N\}$

Algorithm 11: Pseudopolynomial algorithm, where $C_{max}(S_1)(e)$ (resp. $C_{max}(S_2)(e)$) denotes the first (resp. third) element of 3-tuple e.

Theorem 6. *Algorithm 11 returns the payoff of an optimal schedule.*

Proof. Recall that jobs are numbered w.r.t. ERD order: $J_1 \prec_{ERD} \cdots \prec_{ERD} J_N$. Moreover, recall that at each step j, three functions g_1, g_2, g_3 were defined in Section 4.2.1, taking as argument a 4-tuple. We define the corresponding functions q_1, q_2, q_3, such that, for $i \in \{1,2,3\}$: $g_i(C_{max}(S_1), B(S_2), C_{max}(S_2), v, j) = (C_{max}(S'_1), B(S'_2), C_{max}(S'_2), v') \Leftrightarrow q_i(\langle C_{max}(S_1), B(S_2), C_{max}(S_2)\rangle, v, j) = (\langle C_{max}(S'_1), B(S'_2), C_{max}(S'_2)\rangle, v')$. Finally, let $\mathcal{Q}_0 = \{(\langle 0, D_2, 0\rangle, 0)\}$ and, for any $j \in \{1, \ldots, N\}$, let $Q_j = \cup_{(e,v)\in\mathcal{Q}_{j-1}}(q_1(e,v,j) \cup q_2(e,v,j) \cup q_3(e,v,j))$.

Algorithm 11 constructs exactly the sets $\mathcal{Q}_j$, $j = \{1, \ldots, N\}$, except for the similar pairs: for each subset of similar pairs, only one of the pairs with the maximal value of v is kept. Indeed, line 3 clearly adds $\{q_3(e,v,j) : (e,v) \in \mathcal{Q}_{j-1}\}$ to $\mathcal{Q}_j$, lines 4-8 add $\{q_1(e,v,j) : (e,v) \in \mathcal{Q}_{j-1}\}$ to $\mathcal{Q}_j$ and lines 9-13 add $\{q_2(e,v,j) : (e,v) \in \mathcal{Q}_{j-1}\}$ to $\mathcal{Q}_j$; all these additions are performed while observing the avoidance of similar pairs. We show next that $\mathcal{Q}_N$, when constructed without avoiding similar pairs, contains some pair corresponding to an optimal schedule. Therefore, since the removal of similar pairs from $\mathcal{Q}_j$, $j = 1, \ldots, N$, clearly does not prevent to have at least one pair corresponding to an optimal schedule in $\mathcal{Q}_N$, that will prove that Algorithm 11 returns an optimal payoff.

NOTATION. For any ERD-schedule S where S_2 is a block, we denote by $f(S)$ the pair (e,v) corresponding to S's 3-tuple and payoff respectively.

Let S^* be an optimal ERD-schedule where S_2^* is a block. Let us denote by $\{J_{i_1}, \ldots, J_{i_l}\}$, $i_1 < i_2 < \cdots < i_l$ (i.e. $J_{i_1} \prec_{ERD} \cdots \prec_{ERD} J_{i_l}$), the set of jobs of S^* that complete at or before D_2. For $h = 1, .., l$, let S^{i_h} be the schedule that satisfies all the following conditions, with minimum $C_{max}(S_1^{i_h})$ and $C_{max}(S_2^{i_h})$ (among all the schedules satisfying the same conditions):

1. S^{i_h} schedules all the jobs $J_{i_1}, \ldots, J_{i_h}$ before D_2, and all the other jobs after D_2
2. if $C_{i_x}(S^*) \leq D_1$, then $C_{i_x}(S^{i_h}) \leq D_1$, $\forall x \in \{1, \ldots, h\}$
3. if $D_1 < C_{i_x}(S^*) \leq D_2$, then $D_1 < C_{i_x}(S^{i_h}) \leq D_2$, $\forall x \in \{1, \ldots, h\}$
4. S^{i_h} is an ERD-schedule where $S_2^{i_h}$ is a block

We show by induction that, for every $h \in \{1, \ldots, l\}$, $f(S^{i_h}) \in \mathcal{Q}_{i_h}$, which implies $f(S^{i_l}) \in \mathcal{Q}_{i_l} \subseteq \mathcal{Q}_N$, which indeed proves the theorem, since $v(S^{i_l}) = v(S^*)$.

First step of the induction. The only 3-tuple of set $\mathcal{Q}_0$ corresponds to a schedule with no jobs before D_2. If $i_1 > 1$, for all $i \in \{1, \ldots, i_1 - 1\}$: $(\langle 0, D_2, 0\rangle, 0) \in \{q_3(e,v,i) : (e,v) \in \mathcal{Q}_{i-1}\} \subseteq \mathcal{Q}_i$, by definition of $\mathcal{Q}_i$. Then, there are two cases.

If $C_{i_1}(S^*) \leq D_1$, then, in S^{i_1}, J_{i_1} completes at its earliest possible completion time $r_{i_1} + p_{i_1}$, and all the other jobs are executed after D_2. Therefore, $f(S^{i_1}) = (\langle r_{i_1} + p_{i_1}, D_2, 0\rangle, 2) = q_1(\langle 0, D_2, 0\rangle, 0, i_1) \in \{q_1(e,v,i_1) : (e,v) \in \mathcal{Q}_{i_1-1}\} \subseteq \mathcal{Q}_{i_1}$.

Otherwise, if $D_1 < C_{i_1}(S^*) \leq D_2$, then, in S^{i_1}, job J_{i_1} completes at its earliest possible completion time into $]D_1, D_2]$: $\max(r_{i_1}, b_{i_1}) + p_{i_1}$. Therefore, $f(S^{i_1}) = (\langle 0, \max(r_{i_1}, b_{i_1}), \max(r_{i_1}, b_{i_1}) + p_{i_1} \rangle, 1) = q_2(\langle 0, D_2, 0 \rangle, 0, i_1) \in \{q_2(e, v, i_1) : (e, v) \in \mathcal{Q}_{i_1-1}\} \subseteq \mathcal{Q}_{i_1}$.

General step of the induction. Assume now that $f(S^{i_{j-1}}) \in \mathcal{Q}_{i_{j-1}}$. There are two cases.

If $C_{i_j}(S^*) \leq D_1$, then, in S^{i_j}, job J_{i_j} must start after both $C_{max}(S_1^{i_{j-1}})$ and r_{i_j}, in order to satisfy condition 4 and to maintain feasibility. So, the earliest possible starting time of job J_{i_j} is $\max(r_{i_j}, C_{max}(S_1^{i_{j-1}}))$. Moreover, $S_2^{i_j}$ must start not earlier than both the starting time of $S_2^{i_{j-1}}$ (since $C_{max}(S_2^{i_{j-1}})$ is minimal) and the completion time of J_{i_j} in $S_1^{i_j}$ (to avoid overlaps). Hence, if $S_2^{i_{j-1}}$ starts before the completion time of J_{i_j} in $S_1^{i_j}$ (i.e. $\max(r_{i_j}, C_{max}(S_1^{i_{j-1}})) + p_{i_j} > B(S_2^{i_{j-1}})$), then $S_2^{i_j}$ must start exactly at the completion time of J_{i_j} in S^{i_j}. Otherwise, $S_2^{i_j}$ must start at $B(S_2^{i_{j-1}})$. In both cases, $C_{i_j}(S^{i_j}) \leq D_1$, since $C_{i_j}(S^*) \leq D_1$, and $S_1^{i_j}$ is a left-shifted subschedule of S^*. Hence q_2 adds $f(S^{i_j})$ to $\mathcal{Q}_{i_j}$.

Otherwise, if $D_1 < C_{i_j}(S^*) \leq D_2$, then the earliest possible starting time of J_{i_j} in S^{i_j} is clearly $\max(b_{i_j}, C_{max}(S_1^{i_{j-1}}))$ if $S_2^{i_{j-1}}$ is empty, otherwise it is $\max(r_{i_j}, C_{max}(S_2^{i_{j-1}}))$. Function q_2 places the job precisely at that date and, in order for $S_2^{i_j}$ to remain a block, it right-shifts all its jobs so that $S_2^{i_j}$ contains no idle time. $C_{i_j}(S^{i_j}) \leq D_2$, since $C_{i_j}(S^*) \leq D_2$, and $S_2^{i_j}$ is a left-shifted block subschedule of S^*. Hence, $f(S^{i_j}) \in \mathcal{Q}_{i_j}$. □

Proposition 10. *Algorithm 11 computes the optimal payoff of an instance in pseudopolynomial time ($O(N(D_1(D_2)^2) + N \log N)$).*

Proof. First, observe that, by lines 8 and 13, it is impossible that some $\mathcal{Q}_j$ contains two pairs (e, v) and (e', v') with $e = e'$. So the number of elements in each $\mathcal{Q}_j$ is bounded by the number of possible 3-tuples. Clearly, $C_{max}(S_1)$ can only range from 1 to D_1 and $C_{max}(S_2)$ from $D_1 + 1$ to D_2. $B(S_2)$ can only range from $b = \min_{j=1,\ldots,N} b_j^{min}$ to $D_2 - 1$. Overall, the number of states in each $\mathcal{Q}_j$ is bounded by $X = (D_1) \times (D_2 - D_1) \times (D_2 - b)$. In each for loop of lines 2–13, we first copy $\mathcal{Q}_{j-1}$ into $\mathcal{Q}_j$, hence inducing a complexity of $O(X)$, then for each loop of lines 4–8, we try to reinsert job J_j into S_1 for each 3-tuple of $\mathcal{Q}_{j-1}$, hence inducing an overall complexity of $O(X)$ to create states e on line 6 and, by storing $\mathcal{Q}_j$ as an array or a hash table, a complexity of $O(X)$ is induced to update $\mathcal{Q}_j$ on line 8. The same complexity clearly applies for the foreach loop of lines 9–13. So, overall, the complexity of executing lines 3–13 is $O(X) = O(D_1(D_2)^2)$. The for loop of lines 2–13 is executed N times. Finally, sorting the jobs in ERD order can be achieved in $O(N \log N)$. Overall, the complexity of Algorithm 11 is $O(N(D_1(D_2)^2) + N \log N)$. □

4.4.3 The approximation algorithm

We prove in this section the result of Theorem 5.

Theorem 5. $U_2 - |\mathcal{J}(S^l_{1,2})| \leq 1$.

First, we prove the intermediate results of Lemmas 4, 5 and 6, concerning (partial) schedules S^{D_2} and $S^l_{1,2}$. Then, we introduce another partial schedule, S^{mid}, and give its features. Finally, we prove the main result of Theorem 5.

We suppose that the jobs of $\mathcal{J}^{all}$ are reindexed in ERD order: $J_1 \prec_{ERD} \cdots \prec_{ERD} J_N$. Since we will often refer to Properties 1, 2, 3, 4 presented in Chapter 2, we recall them here, for the sake of readability.

Property 1. *Given a set $\mathcal{J}$ of jobs and an interval $[I_{low}, I_{up}]$, the SDD-algorithm produces a partial schedule that schedules the maximal number $N(\mathcal{J}, I_{low}, I_{up})$ of jobs of $\mathcal{J}$ between I_{low} and I_{up}.*

Property 2. *Given a set $\mathcal{J}$ of jobs and an interval $[I_{low}, I_{up}]$, the SDD-algorithm produces a partial schedule with the shortest processing time $p(\mathcal{J}, I_{low}, I_{up})$ among all the feasible partial schedules of $N(\mathcal{J}, I_{low}, I_{up})$ jobs of $\mathcal{J}$ between I_{low} and I_{up}.*

Property 3. *The SDD-algorithm produces a partial schedule where the jobs are ordered w.r.t. ERD order.*

Property 4. *When SDD-algorithm removes a job from the current sequence, it chooses the job with the longest processing time, and breaks ties by choosing the job with the highest ranking in ERD order (i.e. that follows the other jobs with same longest processing time, in ERD order).*

The (partial) schedules that will be useful for the proof are the following. Notice that, by Property 3, the jobs are scheduled in ERD order in S^{D_2}, in S^l_1 and in S^l_2.

- $S^{D_2} = SDD(\mathcal{J}^{all}, 0, D_2)$
- $S^l_1 = \text{left-shift}(SDD(\mathcal{J}^{all}, 0, D_1))$
- $S^l_2 = SDD(\mathcal{J}^{all} \backslash \mathcal{J}(S^l_1), C_{max}(S^l_1), D_2)$
- $S^l_{1,2} = S^l_1.S^l_2$

NOTATION. For any job $J_i \in \mathcal{J}^{all}$, we denote by $\mathcal{J}_i^+$ (resp. $\mathcal{J}_i^-$) the set of jobs of $\mathcal{J}^{all}$ that follow J_i in ERD order, i.e with greater indices (resp. that precede J_i in ERD order, i.e with smaller indices).

In Lemma 4, we show that any job J_l of S^l_2 such that all the following jobs in S^l_2 belong to S^{D_2} has a shorter or equal processing time than any job J_d of S^{D_2} that is not in $S^l_{1,2}$. The result holds, whether or not J_l belongs to S^{D_2}.

Lemma 4. *For any pair of jobs J_d, J_l such that:*

- $J_d \in \mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2})$, *and*
- $J_l \in \mathcal{J}(S^l_2)$, *and*
- $\mathcal{J}^+_l \cap \mathcal{J}(S^l_2) \subseteq \mathcal{J}(S^{D_2})$,

we have: $p_d \geq p_l$.

Proof. By contradiction, suppose that $p_d < p_l$. By Property 3, the jobs of $\mathcal{J}^+_l \cap \mathcal{J}(S^l_2)$ are exactly the jobs scheduled after J_l in S^l_2. Let b_l be the starting time of J_l in S^l_2: $b_l = D_2 - p(\mathcal{J}^+_l \cap \mathcal{J}(S^l_2)) - p_l$ (see Figure 4.15a). Let us now consider S^{D_2}, and suppose that we remove from it all the jobs, except J_d and the jobs of $\mathcal{J}^+_l \cap \mathcal{J}(S^l_2)$. We obtain a feasible (partial) schedule (see Figure 4.15b). If the jobs of the obtained schedule are then right-shifted so that the last one completes at D_2, then the schedule remains feasible. Moreover, the obtained schedule S includes J_d and the jobs of $\mathcal{J}^+_l \cap \mathcal{J}(S^l_2)$, and starts at $b_l + p_l - p_d > b_l$, since $p_d < p_l$. Therefore, there exists a feasible partial schedule $\tilde{S}^l_2$ (see Figure 4.15c) without idle times and that completes at D_2, scheduling the following jobs: first the jobs of $\mathcal{J}^-_l \cap \mathcal{J}(S^l_2)$ in the same order as in S^l_2, followed by S. Moreover, $\tilde{S}^l_2$ schedules only jobs of $\mathcal{J}^{all}\backslash\mathcal{J}(S^l_1)$, between $C_{max}(S^l_1)$ and D_2; and since $\mathcal{J}(\tilde{S}^l_2) = (\mathcal{J}(S^l_2)\backslash\{J_l\}) \cup \{J_d\}$, we have that $|\mathcal{J}(\tilde{S}^l_2)| = |\mathcal{J}(S^l_2)|$, and that $p(\tilde{S}^l_2) < p(S^l_2)$, since $p_d < p_l$. This is in contradiction with Property 2 on S^l_2. □

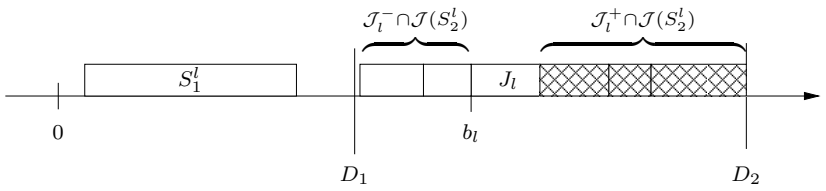
In Lemma 5, we show that all the jobs of S^l_2 are scheduled in S^{D_2}. We first show that, if there exists a job J_l in $\mathcal{J}(S^l_2)\backslash\mathcal{J}(S^{D_2})$, then $J_d \prec_{ERD} J_l$ and $p_d > p_l$, for all $J_d \in \mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2})$. Then, we show that $\tilde{S}^{D_2} = SDD(\mathcal{J}(S^{D_2}) \cup \{J_l\}, 0, D_2)$ is equal to S^{D_2}. Finally, we show that J_l belongs to $\tilde{S}^{D_2}$. Hence, J_l belongs to S^{D_2}, which is in contradiction with the initial assumption.

In order to show that J_l belongs to $\tilde{S}^{D_2}$, we will examine the way SDD-algorithm constructs a schedule. We recall here SDD-algorithm (cf. Algorithm 12, p. 98), where, for the sake of coherence with the rest of the proof, the jobs are reindexed in ERD order (in SDD-algorithm as presented in Chapter 2, the jobs were reindexed in inverse ERD order).

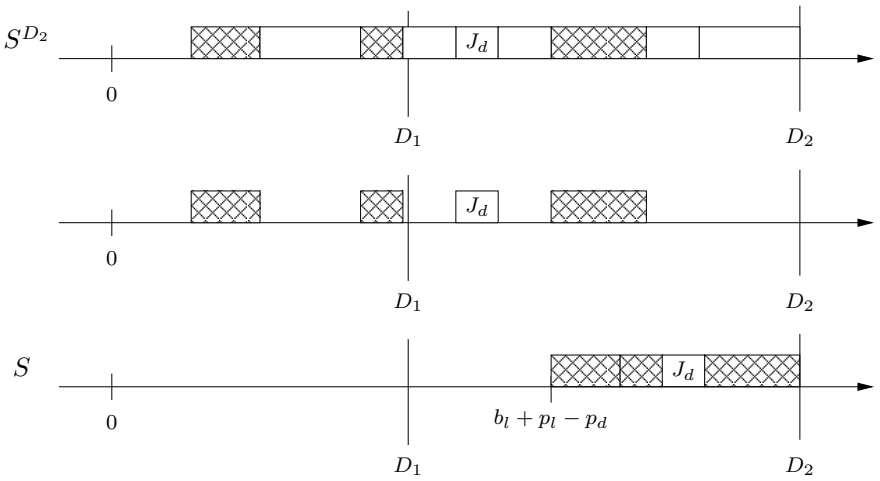
NOTATION. Recall that, for any sequence Γ, $sched(\Gamma)$ denotes the schedule that schedules the jobs of Γ, in the order given by Γ, without idle times and with the last job completing at I_{up}.

Lemma 5. $\mathcal{J}(S^l_2) \subseteq \mathcal{J}(S^{D_2})$.

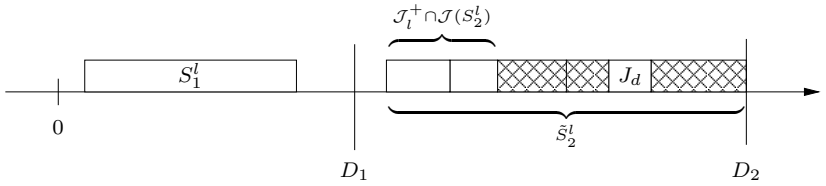
Proof. By contradiction, let us suppose that there exists at least one job in $\mathcal{J}(S^l_2)\backslash\mathcal{J}(S^{D_2})$. Then, there exists at least one job in $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2})$, since $|\mathcal{J}(S^{D_2})| \geq |\mathcal{J}(S^l_{1,2})|$, from Property 1 on S^{D_2}.



(a) (Partial) schedule $S_{1,2}^l$.



(b) From top to bottom: (partial) schedule S^{D_2}, an intermediate partial schedule, partial schedule S.



(c) (Partial) schedule $S_1^l.\tilde{S}_2^l$.

Figure 4.15: Examples for Lemma 4.

Let J_l be the last scheduled job in S_2^l among the jobs of $\mathcal{J}(S_2^l)\backslash\mathcal{J}(S^{D_2})$. We have: $J_l \in \mathcal{J}(S_2^l)$, and all the jobs that succeed J_l in S_2^l belong to S^{D_2}: $\mathcal{J}_l^+ \cap \mathcal{J}(S_2^l) \subseteq \mathcal{J}(S^{D_2})$. Then, by Lemma 4, for any d such that $J_d \in \mathcal{J}(S^{D_2})\backslash\mathcal{J}(S_{1,2}^l)$: $p_d \geq p_l$.

Let us show that $d < l$ (i.e. that $J_d \prec_{ERD} J_l$). By contradiction, suppose that $d \geq l$. We cannot have $d = l$ because $J_l \in \mathcal{J}(S_{1,2}^l)$ while $J_d \notin \mathcal{J}(S_{1,2}^l)$. So, $d > l$. Then, since $d > l$ (which implies $r_d \geq r_l$) and $p_d \geq p_l$, J_d can be replaced by J_l in S^{D_2}, which yields a schedule $\widehat{S}^{D_2}$ with as many jobs as in S^{D_2} (see Figure 4.16). If $p_d > p_l$, then $p(\widehat{S}^{D_2}) < p(S^{D_2})$, which is in contradiction with Property 2 on S^{D_2}. Otherwise, if $p_d = p_l$, $p(\widehat{S}^{D_2}) = p(S^{D_2})$ but $\widehat{S}^{D_2}$ contains a job with a smaller index (J_l, instead of J_d in S^{D_2}), which is in contradiction with Property 4 on S^{D_2}.

Input: $\mathcal{J}$, I_{low}, I_{up}
Output: S
$\Gamma_{i_{|\mathcal{J}|+1}} \leftarrow \emptyset$, $t \leftarrow I_{up}$
for $m = |\mathcal{J}|$ **to** 1 **do**
  $\Gamma_{i_m} \leftarrow J_{i_m}.\Gamma_{i_{m+1}}$
  $t \leftarrow t - p_{i_m}$
  if $t < \max\{r_{i_m}, I_{low}\}$ **then**
    $q \leftarrow \min\{l | p_{i_l} = \max\{p_{i_j} | J_{i_j} \in \mathcal{J}(\Gamma_{i_m})\}\}$
    $\Gamma_{i_m} \leftarrow \Gamma_{i_m} \backslash J_{i_q}$
    $t \leftarrow t + p_{i_q}$
return $S \leftarrow sched(\Gamma_{i_1})$

Algorithm 12: SDD-algorithm.

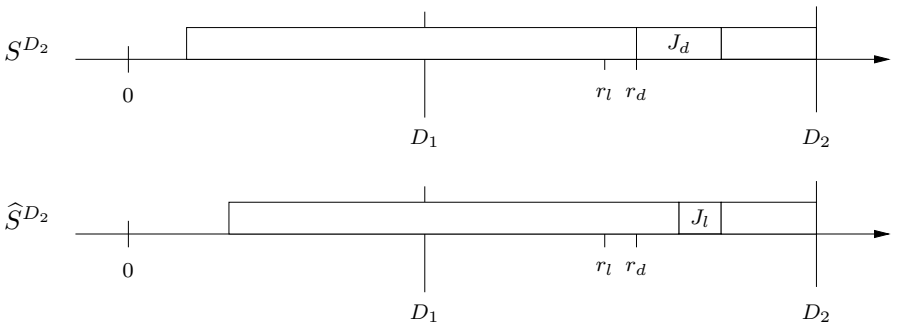


Figure 4.16: Example 1 for Lemma 5 when $p_d > p_l$.

So, we have: $p_d \geq p_l$ and $d < l$ (which implies $r_d \leq r_l$). Let us show that $p_d > p_l$. By contradiction, suppose that $p_d = p_l$. Then, J_l can be replaced by J_d in S_2^l (see Figure 4.17), which yields a schedule with as many jobs as S_2^l, whose total processing time is $p(S_2^l)$, and containing a job with a smaller index. This is in contradiction with Property 4 on S_2^l. Thus, $p_d > p_l$.

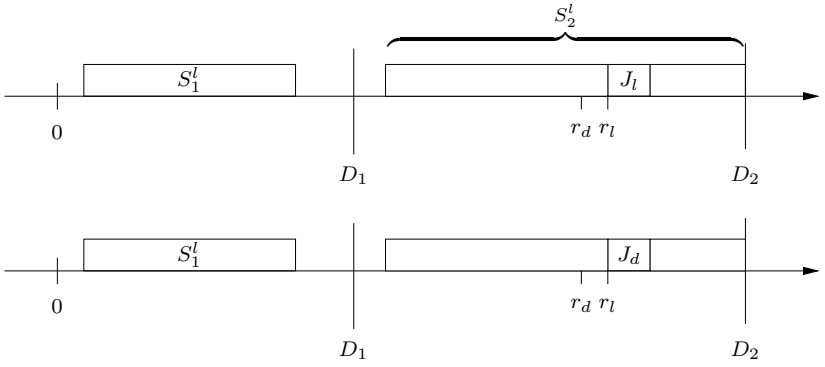


Figure 4.17: Example 2 for Lemma 5.

Therefore, we have, for all $J_d \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_{1,2})$:

- $d < l$ $\quad (\alpha)$
- $p_d > p_l$ $\quad (\beta)$

Figure 4.18 shows how J_l and the other jobs are distributed in $S^l_{1,2}$. From Property 3 on S^l_1 and S^l_2, the jobs of S^l_1 (respectively S^l_2) are scheduled in ERD order.

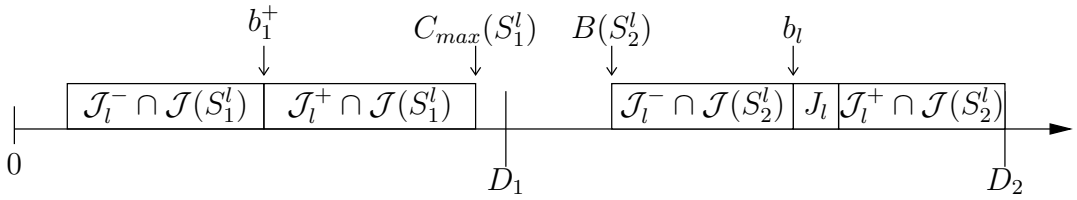


Figure 4.18: The position of J_l and the other jobs in $S^l_{1,2}$.

NOTATION. b_1^+ denotes the starting time of the first scheduled job in S^l_1 among the jobs of $\mathcal{J}_l^+$. b_l denotes the starting time of J_l in S^l_2.

In order to get the contradiction that will prove the lemma, let us consider the following set of jobs: $\mathcal{J}(S^{D_2}) \cup \{J_l\}$. We call $\tilde{S}^{D_2}$ the partial schedule obtained by applying the SDD-algorithm on the jobs of $\mathcal{J}(S^{D_2}) \cup \{J_l\}$ between 0 and D_2: $\tilde{S}^{D_2} = SDD(\mathcal{J}(S^{D_2}) \cup \{J_l\}, 0, D_2)$. We first show that $\tilde{S}^{D_2} = S^{D_2}$. Then, we will show that $J_l \in \mathcal{J}(\tilde{S}^{D_2})$, which is in contradiction with the hypothesis $J_l \notin \mathcal{J}(S^{D_2})$.

Since the jobs of $\mathcal{J}(S^{D_2}) \cup \{J_l\}$ are a subset of the total set of jobs $\mathcal{J}^{all}$, no more of $|\mathcal{J}(S^{D_2})|$ of them can be scheduled between 0 and D_2, in $\tilde{S}^{D_2}$, from Property 1 on S^{D_2}. Moreover, at least $|\mathcal{J}(S^{D_2})|$ of them are scheduled in $\tilde{S}^{D_2}$ (since the jobs of $\mathcal{J}(S^{D_2})$ can be scheduled between 0 and D_2). So, $|\mathcal{J}(\tilde{S}^{D_2})| = |\mathcal{J}(S^{D_2})|$. Let us prove that $\mathcal{J}(\tilde{S}^{D_2}) = \mathcal{J}(S^{D_2})$. This will imply $\tilde{S}^{D_2} = S^{D_2}$, since both schedules complete at D_2 without idle times, and they both schedule the jobs in ERD order (i.e. increasing indices). Suppose by contradiction that $\mathcal{J}(\tilde{S}^{D_2}) \neq \mathcal{J}(S^{D_2})$. Since J_l is the only job among those of $\mathcal{J}(S^{D_2}) \cup \{J_l\}$ that is not in $\mathcal{J}(S^{D_2})$, and since $|\mathcal{J}(\tilde{S}^{D_2})| = |\mathcal{J}(S^{D_2})|$, we deduce that $J_l \in \mathcal{J}(\tilde{S}^{D_2})$ and that there exists one job in $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(\tilde{S}^{D_2})$. Moreover, from Properties 2 and 4 on $\tilde{S}^{D_2}$, one of the following assertions holds:

- $p(\mathcal{J}(\tilde{S}^{D_2})) < p(\mathcal{J}(S^{D_2}))$ (which is in contradiction with Property 2 on S^{D_2}), or
- $p(\mathcal{J}(\tilde{S}^{D_2})) = p(\mathcal{J}(S^{D_2}))$ and l is smaller than the index of the job of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(\tilde{S}^{D_2})$ (which is in contradiction with Property 4 on S^{D_2})

Therefore, $\tilde{S}^{D_2} = S^{D_2}$.

Now, by examining the construction of $\tilde{S}^{D_2}$, we show that $J_l \in \mathcal{J}(\tilde{S}^{D_2})$, which is in contradiction with the result above. Let J_{l^+} be the job of $\mathcal{J}(S^{D_2}) \cup \{J_l\}$ that immediately follows J_l in ERD order.

When constructing $\tilde{S}^{D_2}$ with the SDD-algorithm: $SDD(\mathcal{J}(S^{D_2}) \cup \{J_l\}, 0, D_2)$, the jobs are considered in inverse ERD order (i.e. decreasing indices), and are added (or not) to the current sequence, possibly by removing another job. Any job $J_d \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_{1,2})$ is such that $d < l$ from (α), and is thus considered after J_l when constructing $\tilde{S}^{D_2}$. Therefore, $\mathcal{J}_l^+ \cap \mathcal{J}(S^{D_2}) \subseteq \mathcal{J}_l^+ \cap \mathcal{J}(S^l_{1,2})$. So, when J_l is considered, the jobs of the current sequence Γ_{l^+} (cf. SDD-algorithm p. 98) all belong to $\mathcal{J}_l^+ \cap \mathcal{J}(S^l_{1,2})$.

We now show that J_l can be added to Γ_{l^+} without removing any job from it.

NOTATION. Let B_{l^+} be the starting time of $sched(\Gamma_{l^+})$.

$B_{l^+} = D_2 - p(\mathcal{J}(\Gamma_{l^+})) \geq D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_1)) - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2))$ since $\mathcal{J}(\Gamma_{l^+}) \subseteq \mathcal{J}_l^+ \cap \mathcal{J}(S^l_{1,2})$.

Then, we need to show that the release date of J_l allows it to be scheduled at the beginning of $sched(\Gamma_{l^+})$: $r_l \leq B_{l^+} - p_l$.

If $\mathcal{J}_l^+ \cap \mathcal{J}(S^l_1)$ is empty, then $B_{l^+} \geq D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2)) = b_l + p_l$ (see Figure 4.18). And since $b_l \geq r_l$ (otherwise $S^l_{1,2}$ would not be feasible), we have that $r_l \leq B_{l^+} - p_l$.

Conversely, if $\mathcal{J}_l^+ \cap \mathcal{J}(S^l_1)$ is not empty, then we have: $B_{l^+} \geq D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2)) - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_1))$. Therefore, $B_{l^+} - p_l \geq D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2)) - p_l - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_1)) = b_l - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_1))$ (see Figure 4.18). Moreover, $b_l \geq B(S^l_2) \geq C_{max}(S^l_1) = b_1^+ + p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_1))$. From the inequalities above: $B_{l^+} - p_l \geq b_1^+$. Furthermore, $r_l \leq b_1^+$, because the job that starts at time b_1^+ in $S^l_{1,2}$ has a release date greater than or equal to r_l, since it belongs to $\mathcal{J}_l^+$ and $S^l_{1,2}$ is feasible. From $B_{l^+} - p_l \geq b_1^+$ and $r_l \leq b_1^+$, we deduce: $r_l \leq B_{l^+} - p_l$.

Therefore, J_l can be added to Γ_{l^+} without removing any job.

However, J_l does not belong to $\tilde{S}^{D_2}$ since $\tilde{S}^{D_2} = S^{D_2}$ and $J_l \notin \mathcal{J}(S^{D_2})$: thus, by the SDD-algorithm, there exists $J_x \in \mathcal{J}(S^{D_2})$, $x < l$, such that J_l belongs to Γ_{x^+} but not to Γ_x (i.e. J_l is removed from the current partial schedule in order to add J_x).

Suppose that J_x does not belong to $S^l_{1,2}$: $J_x \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_{1,2})$. Therefore, by (β), $p_x > p_l$, which implies that J_l is not removed in order to insert J_x in S^{D_2}. So, $J_x \in \mathcal{J}(S^l_{1,2})$.

Let us show that there exists in Γ_{x^+} at least one job that does not belong to $\mathcal{J}(S^l_{1,2})$. By contradiction, suppose that $\mathcal{J}(\Gamma_{x^+}) \subseteq \mathcal{J}(S^l_{1,2})$. Moreover, $J_x \in \mathcal{J}(S^l_{1,2})$. Let us consider $S^l_{1,2}$, and remove from it all the jobs except those of $\mathcal{J}(\Gamma_{x^+}) \cup \{J_x\}$: we obtain a feasible schedule. If we reorder the jobs by increasing indices (i.e. in ERD order), and right-shift them in order to obtain a unique block that completes in D_2, we still have a feasible schedule (see Figure 4.19). Notice that this schedule is exactly the same obtained by adding J_x at the beginning of $sched(\Gamma_{x^+})$. Hence, no job of Γ_{x^+} is removed in order to add J_x, and in particular J_l is not removed from Γ_{x^+}. Therefore, $J_l \in \mathcal{J}(\tilde{S}^{D_2}) = \mathcal{J}(S^{D_2})$,

which is in contradiction with the hypothesis $J_l \notin \mathcal{J}(S^{D_2})$. Therefore, there exists at least one job $J_d \in \mathcal{J}(\Gamma_{x^+}) \backslash \mathcal{J}(S^l_{1,2})$.

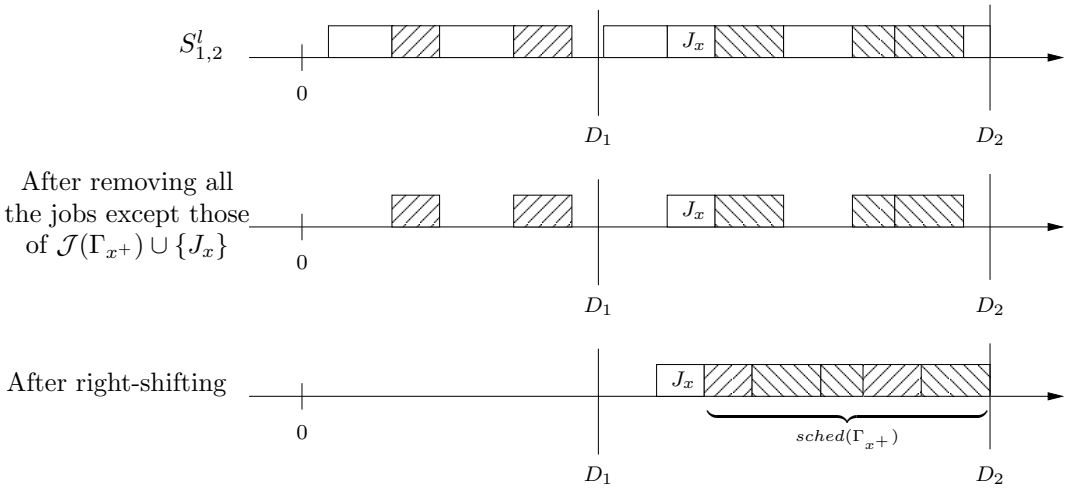


Figure 4.19: Example 4 for Lemma 5: the striped jobs are the jobs of $\mathcal{J}(\Gamma_{x^+})$.

From (β), $p_d > p_l$. Hence, there exists at least one job in Γ_{x^+} that has a longer processing time than J_l. Therefore, J_l cannot be removed in order to add J_x. Consequently, $J_l \in \mathcal{J}(\tilde{S}^{D_2}) = \mathcal{J}(S^{D_2})$, which is in contradiction with the hypothesis $J_l \notin \mathcal{J}(S^{D_2})$. □

Let us introduce the following notation, necessary for Lemma 6.

NOTATION. We set: $y = B(S^l_2) - C_{max}(S^l_1)$ (see Figure 4.20).

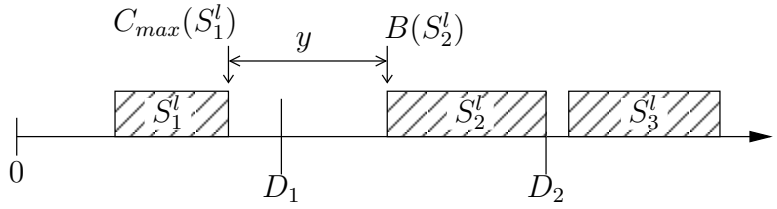
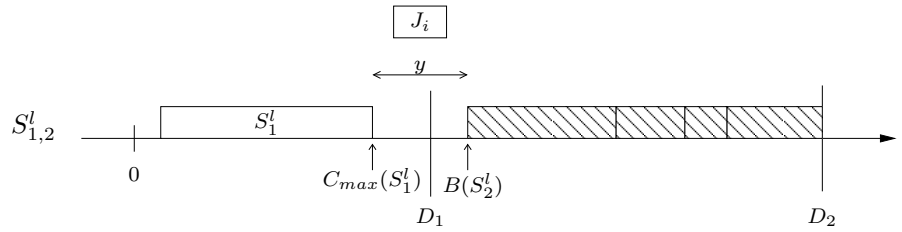


Figure 4.20: Illustration of $y = B(S^l_2) - C_{max}(S^l_1)$.

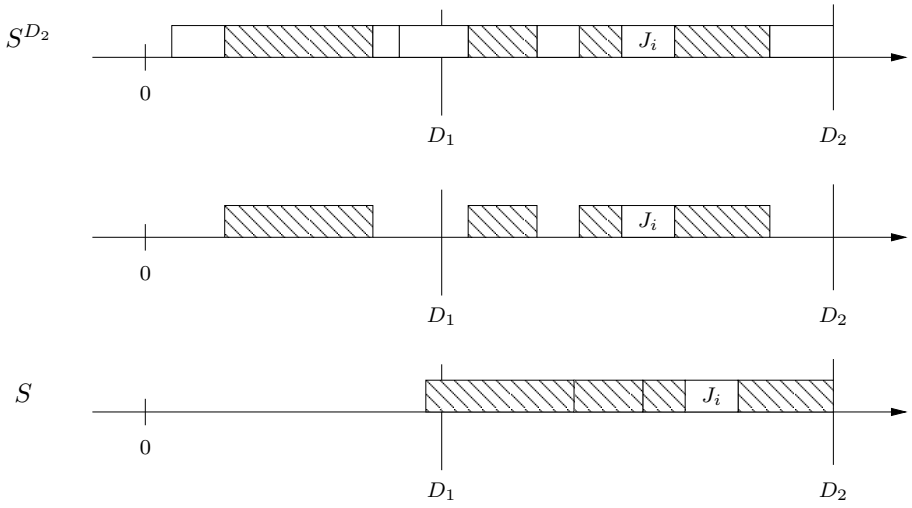
Lemma 6. *For any job J_i of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_{1,2})$: $p_i > y$.*

Proof. By contradiction, suppose that $p_i \leq y$. Let us consider S^{D_2}, and suppose that we remove from it all the jobs, except J_i and the jobs of S^l_2 ($\mathcal{J}(S^l_2) \subseteq \mathcal{J}(S^{D_2})$ by Lemma 5). We obtain a feasible (partial) schedule. If the remaining jobs are then right-shifted so that the last one completes at D_2, the schedule still remains feasible. Moreover, the

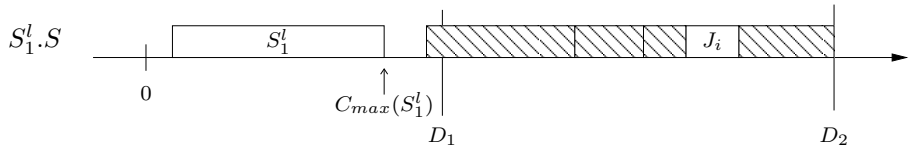
obtained schedule S includes J_i and the jobs of S_2^l, and starts at $B(S_2^l) - p_i \geq C_{max}(S_1^l)$, because $p_i \leq y$ (see Figure 4.21). Therefore, S_2^l can be replaced by S in $S_{1,2}^l$, and $|\mathcal{J}(S)| > |\mathcal{J}(S_2^l)|$. This leads to a contradiction with Property 1 on S_2^l. □



(a) (Partial) schedule $S_{1,2}^l$, and job J_i, which is not in the schedule, and whose processing time is shorter than y.



(b) From top to bottom: (partial) schedule S^{D_2}, an intermediate partial schedule, partial schedule S.



(c) (Partial) schedule $S_1^l.S$.

Figure 4.21: Example for Lemma 6.

In order to prove the result of Theorem 5, i.e. that $|\mathcal{J}(S^{D_2})| \leq |\mathcal{J}(S_{1,2}^l)| + 1$, we will proceed by contradiction, and suppose, from now on, that $|\mathcal{J}(S^{D_2})| \geq |\mathcal{J}(S_{1,2}^l)| + 2$. We introduce now some necessary notations, before introducing partial schedule S^{mid}.

NOTATIONS. Let $\overline{l}$ be the number of jobs of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2}) = \{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$ ($J_{e_1} \prec_{ERD} \ldots \prec_{ERD} J_{e_{\overline{l}}}$, i.e. $e_1 < \ldots < e_{\overline{l}}$). Let $\overline{d}$ be the number of jobs of $\mathcal{J}(S^l_{1,2})\backslash\mathcal{J}(S^{D_2}) = \mathcal{J}(S^l_1)\backslash\mathcal{J}(S^{D_2})$, by Lemma 5. Since $|\mathcal{J}(S^{D_2})| \geq |\mathcal{J}(S^l_{1,2})| + 2$, we have: $0 \leq \overline{d} \leq \overline{l} - 2$. Let $l_2 = |\mathcal{J}(S^l_2)|$, and $J_{f_1}, \ldots, J_{f_{l_2}}$ be the jobs of S^l_2 ($J_{f_1} \prec_{ERD} \ldots \prec_{ERD} J_{f_{l_2}}$). $J_{f_1}, \ldots, J_{f_{l_2}} \in \mathcal{J}(S^{D_2})$, from Lemma 5. On Figure 4.22 an example is given, where $\overline{l} = 6$, $\overline{d} = 4$ and $l_2 = 5$. The same example will be used in the rest of the proof to illustrate it.

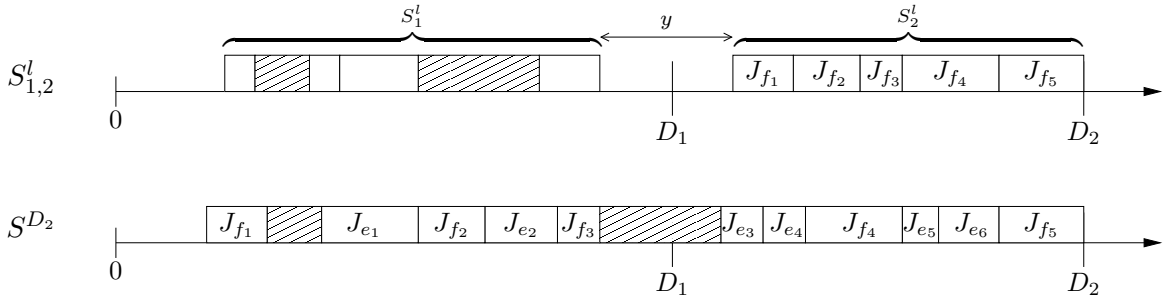


Figure 4.22: On top, the schedule represents $S^l_{1,2}$, while the schedule at bottom represents S^{D_2}, for the same instance. In both schedules, the striped jobs belong to $\mathcal{J}(S^l_1) \cap \mathcal{J}(S^{D_2})$. In S^l_1, the white jobs belong to $\mathcal{J}(S^l_1)\backslash\mathcal{J}(S^{D_2})$. On this example, we have: $\overline{l} = 6$ (the number of J_{e_i} jobs), $\overline{d} = 4$ (the number of white jobs in S^l_1), $l_2 = 5$ (the number of J_{f_j} jobs).

$\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_{1,2})$ and $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2})$ constitute a partition of $\mathcal{J}(S^{D_2})$. Moreover, since $\mathcal{J}(S^l_1)$ and $\mathcal{J}(S^l_2)$ constitute a partition of $\mathcal{J}(S^l_{1,2})$, $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_1)$ and $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_2)$ constitute a partition of $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_{1,2})$. Therefore, a partition of $\mathcal{J}(S^{D_2})$ is constituted of the three sets of jobs:

- $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_1)$,
- $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_2) = \mathcal{J}(S^l_2) = \{J_{f_1}, \ldots, J_{f_{l_2}}\}$,
- $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2}) = \{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$

Then, $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_1) = \{J_{f_1}, \ldots, J_{f_{l_2}}\} \cup \{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$. Hence, $|\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_1)| = \overline{l} + l_2 \geq \overline{d} + 2$.

NOTATIONS. Let G be the set of the $\overline{d}+1$ first jobs of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_1)$ in ERD order. G is also the subset of jobs of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_1)$ with the $\overline{d}+1$ smallest indices[1]. Moreover, the jobs of G are also the $\overline{d}+1$ earliest scheduled jobs of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_1)$ in S^{D_2}, from Property 3 on S^{D_2}. On the example of Figure 4.22, there are $\overline{d} + 1 = 5$ jobs in G: $J_{f_1}, J_{e_1}, J_{f_2}, J_{e_2}, J_{f_3}$.

Let $G^e = G \cap \{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$ and $G^f = G \cap \{J_{f_1}, \ldots, J_{f_{l_2}}\}$.

[1]All along the proof, when we will refer to indices, we will mean the indices of the jobs, and never the subindices. For example, for a given job J_{e_i} we will always refer to the index e_i and never to i. Indeed, indices of job reflect ERD order.

G^e and G^f constitute a partition of G, since $G \subseteq \mathcal{J}(S^{D_2})\backslash\mathcal{J}(S_1^l)$, and the two sets $\{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$ and $\{J_{f_1}, \ldots, J_{f_{l_2}}\}$ are a partition of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S_1^l)$. Therefore, $|G^e| + |G^f| = |G| = \overline{d} + 1$.

Since the jobs of G have the smallest indices among the jobs of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S_1^l)$, we deduce that $\{J_{e_1}, \ldots, J_{e_{\overline{l}}}\} \cap G = \{J_{e_1}, \ldots, J_{e_{|G^e|}}\}$ (since $e_1 < \ldots < e_{|G^e|} < \ldots < e_{\overline{l}}$), and $\{J_{f_1}, \ldots, J_{f_{l_2}}\} \cap G = \{J_{f_1}, \ldots, J_{f_{|G^f|}}\}$ (since $f_1 < \ldots < f_{|G^f|} < \ldots < f_{l_2}$).

On the example of Figure 4.22, $G^e = \{J_{e_1}, J_{e_2}\}$ and $G^f = \{J_{f_1}, J_{f_2}, J_{f_3}\}$.

We introduce now a new partial schedule S^{mid} that is necessary for the proof of Theorem 5. S^{mid} is an intermediate feasible partial schedule between $S_{1,2}^l$ and S^{D_2}: S^{mid} contains all the jobs of $\mathcal{J}(S_{1,2}^l) \cap \mathcal{J}(S^{D_2})$ and some other jobs of S^{D_2}. By the means of S^{mid}, we show that, if $|\mathcal{J}(S^{D_2})| \geq |\mathcal{J}(S_{1,2}^l)| + 2$, then there must be a job of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S_{1,2}^l)$ whose processing time is shorter than $y = B(S_2^l) - C_{max}(S_1^l)$. Thus we find a contradiction with Lemma 6.

The partial schedule S^{mid}

We first show how S^{mid} is constructed, then prove some properties of S^{mid} and S^{D_2}, and finally show that S^{mid} is feasible.

The partial schedule S^{mid} is constructed in the following way, starting from $S_{1,2}^l$ (see Figure 4.23). S^{mid} is composed of two subschedules S_1^{mid} and S_2^{mid}. S_1^{mid} is a left-shifted subschedule obtained from S_1^l by removing the $\overline{d}$ jobs of $\mathcal{J}(S_1^l)\backslash\mathcal{J}(S^{D_2})$, and by adding the $\overline{d}+1$ jobs of G to S_1^l, while maintaining the order of increasing indices (ERD order) guaranteed by Property 3 in S_1^l. S_2^{mid} is the schedule obtained by removing $J_{f_1}, \ldots, J_{f_{|G^f|}}$ from S_2^l.

Remark: Until now the notation S_1 designed a subschedule where all the jobs complete at or before D_1. However, for S^{mid}, we use this notation to design the first (left-shifted) part of the schedule, even if some jobs of S_1^{mid} complete after D_1.

We now introduce some properties of S^{mid}.

Feature 1. *S_1^{mid} is left-shifted.*

Proof. By construction. □

Feature 2. *S_2^{mid} is a right-shifted block that completes at D_2.*

Proof. S_2^{mid} is the second part of the schedule S_2^l, which is also a block that completes at D_2. More precisely, S_2^{mid} is the block that corresponds to the sequence $(J_{f_{|G^f|+1}}, \ldots, J_{f_{l_2}})$ and that completes at D_2 (see Figure 4.23). □

Feature 3. $|\mathcal{J}(S_1^{mid})| = |\mathcal{J}(S_1^l)| + 1$.

Proof. By construction: to obtain S_1^{mid}, $\overline{d}$ jobs were removed from S_1^l, while $\overline{d}+1$ were added (i.e. the $\overline{d}+1$ first jobs of G). □

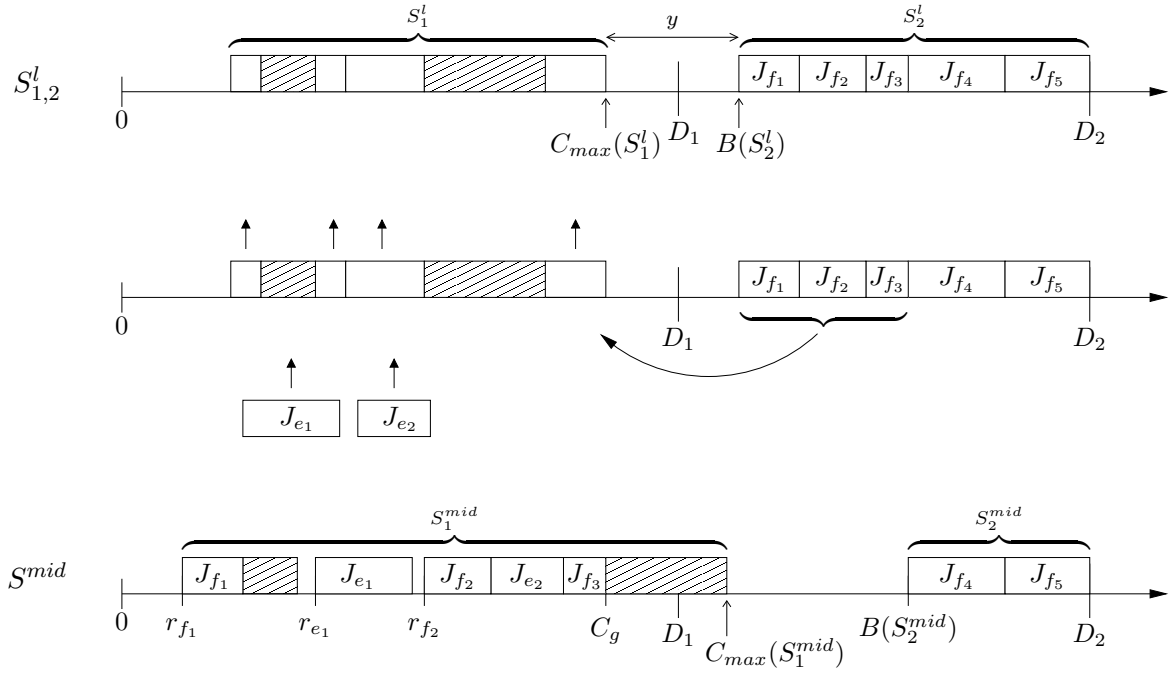


Figure 4.23: Starting from $S_{1,2}^l$, we obtain S^{mid}: the $\overline{d} = 4$ jobs of $\mathcal{J}(S_1^l) \backslash \mathcal{J}(S^{D_2})$ are removed from S_1^l; the $|G^e| = 2$ jobs of $G \cap (\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S_{1,2}^l))$ are added to S_1^l, and the $|G^f| = 3$ jobs of $\mathcal{J}(S_2^l) \cap G$ are moved from S_2^l to S_1^{mid}.

Feature 4. $\mathcal{J}(S^{mid}) \subseteq \mathcal{J}(S^{D_2})$.

Proof. By construction: to construct S^{mid} we removed from $S_{1,2}^l$ the jobs that were not in S^{D_2} and we added the jobs of $G \subseteq \mathcal{J}(S^{D_2})$. □

NOTATION. We denote by $C_{max}(S_1^{mid})$ the completion time of the last job of S_1^{mid}.

Feature 5. $C_{max}(S_1^{mid}) > D_1$.

Proof. By contradiction, assume that $C_{max}(S_1^{mid}) \leq D_1$: since $|\mathcal{J}(S_1^{mid})| = |\mathcal{J}(S_1^l)| + 1$ (cf. Feature 3), it means that $|\mathcal{J}(S_1^l)| + 1$ jobs of $\mathcal{J}^{all}$ can be scheduled in $[0, D_1]$, which is in contradiction with Property 1 on S_1^l. □

NOTATION. Let J_g be the job of G that has the greatest index (i.e. the last job of G in ERD order). It is also the last job of G being scheduled in S_1^{mid} (see Figure 4.23, where $J_g = J_{f_3}$).

Feature 6. *The jobs that are scheduled later than J_g in S_1^{mid} all belong to $\mathcal{J}(S_1^l)$:* $(\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})) \subseteq \mathcal{J}(S_1^l)$.

Proof. By construction of S_1^{mid}, the only jobs of S_1^{mid} that do not belong to S_1^l, are the jobs of G. Since the last scheduled job of G in S_1^{mid} is J_g (by definition of J_g), the jobs that are scheduled later in S_1^{mid} all belong to S_1^l. Moreover, since by construction the jobs in S_1^{mid} are ordered by increasing indices, $\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})$ is actually the set of jobs scheduled after J_g in S_1^{mid}. □

NOTATION. Let $\Gamma_g^{D_2}$ be the sequence of jobs that are scheduled between 0 and $C_g(S^{D_2})$ in S^{D_2}; and Γ_g^{mid} the sequence of jobs that are scheduled between 0 and $C_g(S_1^{mid})$ in S_1^{mid}.

Feature 7. $\Gamma_g^{mid} = \Gamma_g^{D_2}$.

Proof. First, note that $J_g \in G \subseteq \mathcal{J}(S^{D_2})$. From Property 3 on S^{D_2}, $\mathcal{J}(\Gamma_g^{D_2}) = (\mathcal{J}_g^- \cap \mathcal{J}(S^{D_2})) \cup \{J_g\}$. Moreover, since the jobs of S_1^{mid} are ordered by increasing indices (by construction), $\mathcal{J}(\Gamma_g^{mid}) = (\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^-) \cup \{J_g\}$. Thus, showing that $\mathcal{J}(\Gamma_g^{mid}) \cap \mathcal{J}_g^- = \mathcal{J}(\Gamma_g^{D_2}) \cap \mathcal{J}_g^-$ directly implies $\mathcal{J}(\Gamma_g^{mid}) = \mathcal{J}(\Gamma_g^{D_2})$.

By construction of S_1^{mid}, the two sets $\mathcal{J}(S_1^l) \cap \mathcal{J}(S^{D_2})$ and G constitute a partition of $\mathcal{J}(S_1^{mid})$. Therefore, $\mathcal{J}(S_1^l) \cap \mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^-$ and $G \cap \mathcal{J}_g^-$ constitute a partition of $\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^- = \mathcal{J}(\Gamma_g^{mid}) \cap \mathcal{J}_g^-$. Notice that these two sets are also a partition of $\mathcal{J}_g^- \cap \mathcal{J}(S^{D_2}) = \mathcal{J}_g^- \cap \mathcal{J}(\Gamma_g^{D_2})$, since $G \cap \mathcal{J}_g^- = (\mathcal{J}_g^- \cap \mathcal{J}(S^{D_2})) \backslash \mathcal{J}(S_1^l)$. Hence, $\mathcal{J}(\Gamma_g^{mid}) \cap \mathcal{J}_g^- = \mathcal{J}(\Gamma_g^{D_2}) \cap \mathcal{J}_g^-$.

Furthermore, since the jobs in S_1^{mid}, respectively S^{D_2}, are ordered w.r.t. increasing indices (by construction on S_1^{mid} and from Property 3 on S^{D_2}), we deduce that the jobs of $\Gamma_g^{D_2}$ and Γ_g^{mid} are in the same order. □

Feature 8. $\mathcal{J}(S_2^{mid}) \subseteq \mathcal{J}_g^+$.

Proof. $\mathcal{J}(S^{mid}) \subseteq \mathcal{J}(S^{D_2})$ (cf. Feature 4). Then, since the jobs that precede J_g in S^{mid} are the same that the ones preceding J_g in S^{D_2} (cf. Feature 7), we deduce that the jobs that follow J_g in S^{mid} are a subset of the jobs that follow J_g in S^{D_2}. From Property 3 on S^{D_2}, the jobs that follow J_g in S^{D_2} are those of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$. Then, since all the jobs of S_2^{mid} are scheduled after J_g in S^{mid} (because $J_g \in \mathcal{J}(S_1^{mid})$), we deduce that $\mathcal{J}(S_2^{mid}) \subseteq \mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2}) \subseteq \mathcal{J}_g^+$. □

Feature 9. *In S^{D_2}, all the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$ are scheduled after $C_g(S_1^{mid})$.*

Proof. From Property 3 on S^{D_2}, all the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$ are scheduled after J_g.

Let us show that J_g cannot complete earlier than $C_g(S_1^{mid})$ in S^{D_2}. From Feature 7, the sequence of jobs scheduled in S_1^{mid} between 0 and $C_g(S_1^{mid})$ is the same as that of the jobs scheduled in S^{D_2} between 0 and $C_g(S^{D_2})$. Therefore, since S_1^{mid} is left-shifted (cf. Feature 1), J_g cannot complete earlier than $C_g(S_1^{mid})$ in S^{D_2}. Consequently, all the jobs that follow J_g in S^{D_2} are scheduled after $C_g(S_1^{mid})$ in S^{D_2}. □

Remark: In particular, the result of Feature 9 is true for any subset of jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$, and thus for $\mathcal{J}_g^+ \cap \mathcal{J}(S^{mid})$ (cf. Feature 4): all the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{mid})$ are scheduled after $C_g(S_1^{mid})$ in S^{D_2}.

Feature 10. $p(\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})) \leq D_2 - C_g(S_1^{mid})$.

Proof. Since, by Feature 9, all the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$ are scheduled between $C_g(S_1^{mid})$ and D_2 in S^{D_2}, their total processing time must be less than or equal to $D_2 - C_g(S_1^{mid})$. □

Remark 1: In particular, the sum of the processing times of the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{mid})$ is also less than or equal to $D_2 - C_g(S_1^{mid})$.

Remark 2: However, notice that even though $\mathcal{J}(S^{mid}) \cap \mathcal{J}_g^+ \subseteq \mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+$, the sequence of the jobs of $\mathcal{J}(S^{mid}) \cap \mathcal{J}_g^+$ in S^{mid} is not a subsequence of the sequence of jobs of $\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+$ in S^{D_2}. Indeed, the jobs scheduled after J_g in S^{D_2} are scheduled in the order of their increasing indices, while the jobs scheduled after J_g in S^{mid} are partitioned in two groups: the jobs of $\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+$, which are ordered by increasing indices, and the jobs of S_2^{mid}, which are also ordered by increasing indices; but there is no guarantee that the jobs of $(\mathcal{J}(S_1^{mid} \cap \mathcal{J}_g^+)) \cup \mathcal{J}(S_2^{mid})$ are scheduled in the increasing order of their indices in S^{mid}.

Feature 11. *In S_1^{mid}, there are no idle times between any pair of jobs of $\{J_g\} \cup (\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}))$.*

Proof. By contradiction, suppose that there is an idle time in S_1^{mid} between a pair of jobs of $\{J_g\} \cup (\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}))$. Then, since S_1^{mid} is left-shifted (cf. Feature 1) there is at least one job J_q of $\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})$ that starts exactly at its release date r_q. Moreover, $\{J_q\} \cup (\mathcal{J}_q^+ \cap \mathcal{J}(S_1^{mid})) \subseteq \mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}) \subseteq \mathcal{J}(S_1^l)$ (cf. Feature 6). Furthermore, the jobs of $(\mathcal{J}_q^+ \cap \mathcal{J}(S_1^{mid})) \cup \{J_q\}$ are ordered by increasing indices both in S_1^{mid} (by construction) and in S_1^l (by Property 3).

Then, since J_q cannot start earlier than r_q in S_1^l, and since S_1^{mid} is left-shifted (cf. Feature 1), we deduce that S_1^l cannot complete before $C_{max}(S_1^{mid}) > D_1$ (cf. Feature 5), which is clearly a contradiction with the definition of S_1^l (a partial schedule that schedules jobs between 0 and D_1). □

Let us now show that S^{mid} is feasible.

Proposition 11. *S^{mid} is feasible.*

Proof. By construction, each job starts at or after its release date in S^{mid}. Therefore, schedule S^{mid} is unfeasible if and only if $C_{max}(S_1^{mid}) > B(S_2^{mid})$. Let us prove by contradiction that S^{mid} is feasible. We assume that $C_{max}(S_1^{mid}) > B(S_2^{mid})$.

Therefore, if we examine S^{mid} between $C_g(S_1^{mid})$ and D_2 (see Figure 4.24), we see that the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})$ are left-shifted, starting from $C_g(S_1^{mid})$ in a unique block (cf. Feature 11), hence $C_{max}(S_1^{mid}) = C_g(S^{mid}) + p(\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}))$. On the other hand, the jobs of S_2^{mid} are right-shifted, completing at D_2 in a unique block (cf. Feature 2), hence $B(S_2^{mid}) = D_2 - p(\mathcal{J}(S_2^{mid}))$.

$C_{max}(S_1^{mid}) > B(S_2^{mid})$ implies that $p((\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})) \cup \mathcal{J}(S_2^{mid})) = p(\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})) + p(\mathcal{J}(S_2^{mid})) > D_2 - C_g(S_1^{mid})$, which is in contradiction with Feature 10, since $\mathcal{J}(S^{mid}) \subseteq \mathcal{J}(S^{D_2})$ (cf. Feature 4), $\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}) \subseteq \mathcal{J}_g^+$ and $\mathcal{J}(S_2^{mid}) \subseteq \mathcal{J}_g^+$ (cf. Feature 8). Consequently, S^{mid} is feasible. □

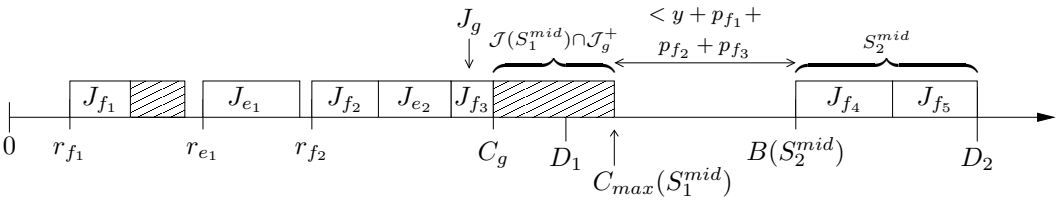


Figure 4.24: S^{mid}.

We can now prove the main result.

Theorem 5. $|\mathcal{J}(S^{D_2})| \leq |\mathcal{J}(S_{1,2}^l)| + 1$

Proof. We have the following partitions of $\mathcal{J}(S^{D_2})$:
$\mathcal{J}(S^{D_2}) = (\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S_{1,2}^l)) \cup (\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S_{1,2}^l))$
$= (\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S_{1,2}^l)) \cup \{J_{e_1}, \ldots, J_{e_{\bar{l}}}\}$
$= (\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S_{1,2}^l)) \cup G^e \cup \{J_{e_{|G^e|+1}}, \ldots, J_{e_{\bar{l}}}\}$
$= \mathcal{J}(S^{mid}) \cup \{J_{e_{|G^e|+1}}, \ldots, J_{e_{\bar{l}}}\}$

Hence, $\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+ = (\mathcal{J}(S^{mid}) \cap \mathcal{J}_g^+) \cup \{J_{e_{|G^e|+1}}, \ldots, J_{e_{\bar{l}}}\}$, since $\{J_{e_{|G^e|+1}}, \ldots, J_{e_{\bar{l}}}\} \subseteq \mathcal{J}_g^+$, since $g < e_{|G^e|+1}$, by construction of G and by definition of g.

Notice that $\mathcal{J}(S^{mid}) \cap \mathcal{J}_g^+ = (\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+) \cup \mathcal{J}(S_2^{mid})$, since $\mathcal{J}(S_2^{mid}) \subseteq \mathcal{J}_g^+$ (cf. Feature 8). Then, since the three sets $\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+$, $\mathcal{J}(S_2^{mid})$ and $\{J_{e_{|G^e|+1}}, \ldots, J_{e_{\bar{l}}}\}$ are disjoint, they constitute a partition of $\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+$. Consequently, $p(\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+) = p(\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+) + p(\mathcal{J}(S_2^{mid})) + p(\{J_{e_{|G^e|+1}}, \ldots, J_{e_{\bar{l}}}\}) \leq D_2 - C_g(S_1^{mid})$, since all the jobs of $\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+$ are scheduled after $C_g(S_1^{mid})$ in S^{D_2} (cf. Feature 9). We have:

- $p(\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+) = C_{max}(S_1^{mid}) - C_g(S_1^{mid})$, since the jobs of $\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+$ are scheduled without idle times between $C_g(S_1^{mid})$ and $C_{max}(S_1^{mid})$ in S^{mid} (cf. Feature 11); and
- $p(\mathcal{J}(S_2^{mid})) = D_2 - B(S_2^{mid})$, since the jobs of $\mathcal{J}(S_2^{mid})$ are scheduled without idle times between $B(S_2^{mid})$ and D_2 in S^{mid} (cf. Feature 2).

We deduce that $p(\{J_{e_{|G^e|+1}}, \ldots, J_{e_{\bar{l}}}\}) \leq D_2 - C_g(S_1^{mid}) - (C_{max}(S_1^{mid}) - C_g(S_1^{mid})) - (D_2 - B(S_2^{mid})) = B(S_2^{mid}) - C_{max}(S_1^{mid})$; which can be written as: $\sum_{i=|G^e|+1}^{\bar{l}} p_{e_i} \leq B(S_2^{mid}) - C_{max}(S_1^{mid})$. Moreover, we have:

- $B(S_2^{mid}) - C_{max}(S_1^{mid}) < B(S_2^{mid}) - C_{max}(S_1^l)$ because $C_{max}(S_1^l) \leq D_1 < C_{max}(S_1^{mid})$ (cf. Feature 5); and
- $B(S_2^{mid}) = B(S_2^l) + \sum_{j=1}^{|G^f|} p_{f_j}$, by construction of S^{mid} (see Figure 4.23); and hence
- $B(S_2^{mid}) - C_{max}(S_1^l) = B(S_2^l) - C_{max}(S_1^l) + \sum_{j=1}^{|G^f|} p_{f_j} = y + \sum_{j=1}^{|G^f|} p_{f_j}$, by definition of y (see Figure 4.23).

From what precedes, $\sum_{i=|G^e|+1}^{\bar{l}} p_{e_i} < y + \sum_{j=1}^{|G^f|} p_{f_j}$, which can be written as:

$$\sum_{i=1}^{\bar{l}-|G^e|} p_{e_{i+|G^e|}} < y + \sum_{j=1}^{|G^f|} p_{f_j}$$

Notice that $\bar{l} - |G^e| > |G^f|$, since $|G^f| = |G| - |G^e| = \overline{d} + 1 - |G^e|$ and $\overline{d} \leq \bar{l} - 2$. Then, for all $i \in \{1, \ldots, |G^f|\}$:

- $J_{e_{i+|G^e|}} \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S_{1,2}^l)$
- $J_{f_i} \in \mathcal{J}(S_2^l)$, and
- $\mathcal{J}_{f_i}^+ \cap \mathcal{J}(S_2^l) \subseteq \mathcal{J}(S^{D_2})$ (since $\mathcal{J}(S_2^l) \subseteq \mathcal{J}(S^{D_2})$ by Lemma 5).

Therefore, by Lemma 4: $\forall i \in \{1, \ldots, |G^f|\}, p_{e_{i+|G^e|}} \geq p_{f_i}$. Hence, $\sum_{i=|G^f|+1}^{\bar{l}-|G^e|} p_{e_{i+|G^e|}} < y$; and this sum has at least one term, since $\bar{l} - |G^e| > |G^f|$. Therefore, $p_{e_{\bar{l}}} < y$, while, from Lemma 6, $p_{e_{\bar{l}}} > y$, which is a contradiction. □

Corollary 1. *Hence, the result of Theorem 4 can be generalized to the case where the two values of the common stepwise payoff function are not* 2 *and* 1*, but any pair of values* v_1*,* v_2 *such that* $v_1 > v_2$*. Then, Algorithm 9 yields a solution whose payoff* v *is such that* $v \geq OPT - v_2$*.*

Proof. The first assertion clearly holds, since the result of Theorem 5 is independent from the payoffs of the jobs. Theorem 5 shows that Algorithm 9 yields a solution where U_1 jobs complete at or before D_1, and at least $U_2 - 1$ jobs complete at or before D_2. Therefore its payoff is at least $(v_1 - v_2) \times U_1 + v_2 \times (U_2 - 1)$, while the upper bound on an optimal payoff is $(v_1 - v_2) \times U_1 + v_2 \times U_2$. Hence the result holds. □

4.5 Conclusion

In this chapter, we presented a specific dominance rule for the single machine problem with two delivery dates. Then, we proposed a pseudopolynomial time exact method, based on dynamic programming which proves the weak NP-hardness of the two delivery dates problem. Since this algorithm remains pseudopolynomial when K is fixed ($K > 2$), we also deduced the weak NP-hardness of the general problem where K is fixed. Finally, we provided a polynomial time algorithm yielding a solution with an absolute performance guarantee.

In the next chapter, we consider a flowshop problem with the same criterion based on delivery dates and cumulative payoffs. For solving the flowshop problem we present some heuristic methods: constructive heuristics, local search methods, and an upper bound to evaluate the quality of the solutions.

Chapter 5

Flowshop problem

(Joint work with Luciana Pessoa[1])

As seen in Section 1.1 (p. 3), the Banctec digitization process is mainly linear, and composed of four main steps. Hence, it can be naturally modeled as a flowshop. So, in order to get closer to the industrial issue, we consider in this chapter a flowshop problem where $\sum_{k=1}^{K} V_k$ must be maximized. However, we would like to point out that the work on this problem is not finished yet, and thus that the results presented here are intended as preliminary results.

We first introduce the studied flowshop problem, then we present some heuristic methods for solving it: constructive heuristics, local search methods, and an upper bound to evaluate the quality of the solutions. Finally, we present some experimental results.

5.1 Definition of the problem and related works

As in the single machine problem, we consider a set of N jobs $\mathcal{J}^{all} = \{J_1, \ldots, J_N\}$, where each job J_i has a release date $r_i \geq 0$. Each job must be sequentially processed on each machine of a set $\{M_1, \ldots, M_m\}$ of m machines, i.e. each job J_i must first be processed on machine M_1, then it is processed on machine M_2, and so on, until machine M_m. The processing of job J_i on a machine must start after the completion of J_i on the preceding machine. The processing time p_{il} of job J_i on machine M_l is strictly greater than zero. We denote by $C_{il}(S)$ the completion time of job J_i on machine M_l in schedule S. For shortness, we denote C_{im} by C_i, $i = 1, \ldots, N$.

The payoff of a schedule is computed in the same way as for the single machine problem (cf. Section 1.2, p. 6), i.e. as the sum of the jobs' payoffs in the schedule, as we

[1]This chapter is the result of a joint work with Luciana Pessoa funded by Dem@tFactory project.

recall below.

Given a set of K delivery dates $D_1, \ldots, D_K$, we set:

- $D_0 = 0$,
- $D_{K+1} = \max(D_K, \max_{i=1,\ldots,N} r_i) + \sum_{i=1}^{N} \sum_{l=1}^{m} p_{il}$,
- $I_k =]D_{k-1}, D_k], k = 1, \ldots, K+1$

The payoff of J_i in a given schedule is: $v(J_i) = K - k + 1$ if $C_i \in I_k$, $k = 1, \ldots, K$.

As a preliminary approach to the flowshop problem, we consider here a permutation flowshop, i.e. each machine processes the jobs in the same order, as in the example of Figure 5.1.

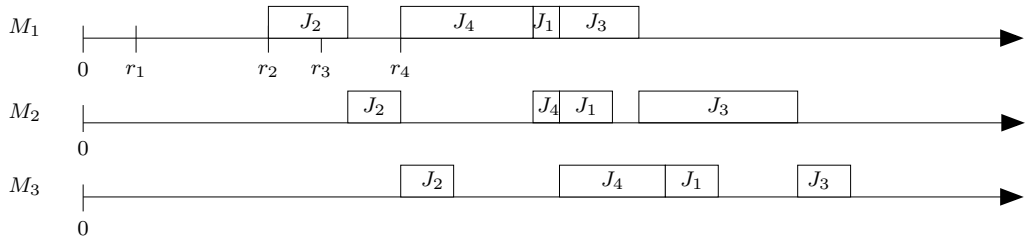


Figure 5.1: A schedule for a permutation flowshop.

Extending the three-field notation of Graham et al. [11], this problem can be denoted as $F|r_j, perm| \sum_{k=1}^{K} V_k$. $F|r_j, perm| \sum_{k=1}^{K} V_k$ is strongly NP-hard, since the single-machine problem $1|r_i| \sum_{k=1}^{K} V_k$ is strongly NP-hard in the general case. Moreover, the special case with two machines, no release dates and a single delivery date $F2|perm|V$ is already NP-hard, as shown by Della Croce et al. [5]. Hence, it seems natural to tackle this problem with heuristic approaches.

Let us introduce the following definition.

Definition 6. *A "left-shifted" schedule is such that each operation of each job starts as soon as possible, after:*

- *its release date, and*
- *the completion time of the preceding operation on the same machine, and*
- *the completion time of the previous operation of the same job on the preceding machine.*

Notice that since the optimization criterion is regular, left-shifted schedules are dominant for $F|r_j, perm| \sum_{k=1}^{K} V_k$. Moreover, a feasible left-shifted schedule can be represented as a sequence, since each machine processes the jobs in the same order. For

instance, the schedule of Figure 5.1 is a left-shifted schedule that can be represented by the sequence (J_2, J_4, J_1, J_3). Hence, this problem can be modeled as an integer program with a positional variables formulation.

According to Della Croce et al. [6] for the problem $F2|perm|\sum C_j$, this formulation is superior to other models based on disjunctive variables and constraints. Before giving the model, let us introduce some necessary notations.

NOTATIONS. We denote by $C_{jl}^{pos}(S)$ (C_{jl}^{pos} when no ambiguity is possible) the completion time of the j-th scheduled job in S on machine M_l in S, $j = 1, \ldots, N$, $l = 1, \ldots, m$. For shortness, C_j^{pos} denotes C_{jm}^{pos}.

The mathematical formulation of $F|r_j, perm|\sum_{k=1}^{K} V_k$ is the following, where the binary decision variable X_{ij} is equal to 1 if job J_i is the j-th job of the sequence and $X_{ij} = 0$ otherwise.

$$\max \sum_{j=1}^{N} \mathcal{F}(C_j^{pos}) \tag{5.1}$$

s.t.

$$\sum_{i=1}^{N} X_{ij} = 1, \quad j = 1, \ldots, N, \tag{5.2}$$

$$\sum_{j=1}^{N} X_{ij} = 1, \quad i = 1, \ldots, N, \tag{5.3}$$

$$C_{j1}^{pos} \geq \sum_{i=1}^{N} (p_{i1} + r_i) X_{ij}, \quad j = 1, \ldots, N, \tag{5.4}$$

$$C_{jl}^{pos} \geq C_{j-1,l}^{pos} + \sum_{i=1}^{N} p_{il} X_{ij}, \quad j = 2, \ldots, N, \quad l = 1, \ldots, m, \tag{5.5}$$

$$C_{j,l+1}^{pos} \geq C_{jl}^{pos} + \sum_{i=1}^{N} p_{i,l+1} X_{ij}, \quad j = 1, \ldots, N, \quad l = 1, \ldots, m-1, \tag{5.6}$$

$$C_{jl}^{pos} \geq 0, \quad j = 1, \ldots, N, \quad l = 1, \ldots, m. \tag{5.7}$$

$$X_{ij} \in \{0, 1\}, \quad i = 1, \ldots, N, \quad j = 1, \ldots, N. \tag{5.8}$$

The objective function is stated in (5.1), where:

$$\mathcal{F}(C_j^{pos}) = \begin{cases} K & \text{if } 0 < C_j^{pos} \leq D_1 \\ \vdots & \\ 2 & \text{if } D_{K-2} < C_j^{pos} \leq D_{K-1} \\ 1 & \text{if } D_{K-1} < C_j^{pos} \leq D_K \\ 0 & \text{if } D_K < C_j^{pos} \end{cases}$$

Constraints (5.2) and (5.3) ensure that each position is attributed to exactly one job and each job is processed at exactly one position. Constraints (5.4) ensure each job J_j to start not earlier than r_j on the first machine. Constraints (5.5) forbid the overlapping of any job with its own predecessor on each machine. Constraints (5.6) establish that a job cannot be processed on a machine before its completion on the preceding machine.

To the best of our knowledge, no flowshop problem with stepwise job cost functions has been studied yet. In the literature, the more closely related problems are flowshops with release dates and regular sum objective functions $\sum_{i=1}^{N} f_i(C_i)$. Rakrouki and Ladhari [19] consider $F2|r_i|\sum C_i$ for which they propose some Branch and Bound methods, solving instances with up to 100 jobs for some classes of instances and 30 jobs for the hardest class of instances. Ladhari and Rakrouki [19] consider the problem $F2|r_i, perm|\sum C_i$, for which they develop lower bounds, heuristics and a genetic algorithm.

Other related problems are flowshop problems with regular sum objective functions $\sum_{i=1}^{N} f_i(C_i)$, but without release dates. Della Croce et al. [6] propose a matheuristic for solving the two machine flowshop problem $F2||\sum C_i$; Della Croce et al. [5] solve $F_2|d_i = d|\sum U_i$ with a Branch and Bound method; and Vallada et al. [33] compare several heuristics for solving $F||\sum T_i$.

In the remaining part of the chapter, we present the proposed heuristic methods, immediately followed by the corresponding experimental results.

In the next section are described the experimental settings.

5.2 Experimental settings

Instances generation We generated 400 test instances for the problem addressed in this work. Each instance contains 100 jobs which must be processed on 2 machines. The processing times are integers drawn randomly from a uniform distribution in the interval $[1, 100]$. The number K of delivery dates varies in $\{1, 2, 3, 5\}$. Delivery dates for each instance are defined as follows. Let C be the makespan of a schedule obtained by using Johnson's rule [2] while ignoring release dates. Then, $D_1 = \lfloor (A \times C)/K \rfloor$, where

A is a parameter with values in $\{0.5, 0.8, 1.0, 1.2\}$. The other delivery dates are $D_k = k \times D_1, 1 < k \leq K$. Release dates are chosen in a similar way as for the single machine problem: each one is picked randomly in one of the intervals $R_k = [D_k, D_k + R \times D_1]$, for $k = 0, \ldots, K$, where $R \in \{0.1, 0.3, 0.5, 0.7, 1.0\}$. The choice of the interval R_k is also random (each with probability $1/K$).

The same instances are used for all the experiments of this chapter. The computational experiments were performed on a 3.16GHz Intel Core2-Duo processor with 4 GB RAM computer. Each run was limited to a single processor. All codes were implemented in C++.

Solution quality evaluation. Two metrics were used to compare the proposed methods.

- *Time* is the CPU time, in seconds.
- *Gap* is the relative gap between the solution payoff provided by the considered heuristic and an upper bound on the optimal payoff of $F|r_i, perm|\sum_{k=1}^{K} V_k$ (described below).

The results of each heuristic are summarized according to K firstly, then to A and finally to R, by giving the average value and the standard deviation, for both *Gap* and *Time*.

An upper bound on the optimal payoff of $F|r_i, perm|\sum_{k=1}^{K} V_k$ is obtained by considering the relaxed problem where the first machine has infinite capacity. This relaxed problem is equivalent to the single machine problem where the release date of job J_i is equal to $r_i + p_{i_1}$ and its processing time is equal to p_{i2}, for all $i \in \{1, \ldots, N\}$. As seen in Section 3.3.2, Algorithm 3 (p. 43) provides an upper bound on the optimal payoff of $1|r_i|\sum_{k=1}^{K} V_k$. Since an upper bound on the payoff of a relaxed problem yields an upper bound on the payoff of the original problem, applying Algorithm 3 on the relaxation of $F|r_i, perm|\sum_{k=1}^{K} V_k$ where the first machine has infinite capacity yields an upper bound on the optimal payoff of $F|r_i, perm|\sum_{k=1}^{K} V_k$.[2]

5.3 Constructive Heuristics

Four constructive heuristics were evaluated for $F|r_j, perm|\sum_{k=1}^{K} V_k$. In this section, we present three heuristics found in the literature, and describe a new greedy constructive heuristic.

[2]In order to evaluate the proposed heuristics, we attempted to compute optimal solutions with the use of Cplex on the positional variables formulation described above. However, after 6 hours computation, the gap was closed only for a few instances. The gaps between the proposed upper bound and these few optimal solutions were all greater than 20%.

The instance described in Table 5.1, with four jobs and two delivery dates on two machines, will be used to illustrate the constructive heuristics.

	r_i	p_{i1}	p_{i2}
J_1	1	2	5
J_2	3	3	3
J_3	4	2	1
J_4	6	2	2
$D_1 = 9;$		$D_2 = 11$	

Table 5.1: An instance of $F2|r_i, perm| \sum V_k$.

ERDH This heuristic produces a left-shifted schedule where jobs are ordered w.r.t. nondecreasing release dates. Potts [24] uses this heuristic for the makespan minimization problem on a two-machine permutation flowshop. The schedule obtained with this heuristic for the instance of Table 5.1 is illustrated in Figure 5.1.

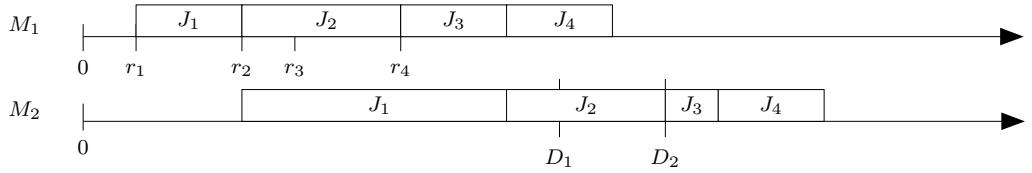


Figure 5.2: The schedule obtained with ERDH heuristic for the instance of Table 5.1, of payoff 3.

ECT This heuristic produces a left-shifted schedule where jobs are ordered w.r.t. their earliest possible completion time $E_i = r_i + \sum_{l=1}^{m} p_{il}$. This $O(NlogN)$ algorithm was proposed by Ladhari and Rakrouki [19] for the minimization of the total completion time on a two-machine flowshop problem with release dates.

For the instance of Table 5.1, we have: $E_1 = 8$, $E_2 = 9$, $E_3 = 7$, $E_4 = 10$. Hence, the schedule constructed with ECT corresponds to the sequence (J_3, J_1, J_2, J_4), and is illustrated in Figure 5.3.

NEH The steps of the NEH-based algorithm are the following:

1. Sort the jobs in the increasing order of $E_i = r_i + \sum_{l=1}^{m} p_{il}$
2. Consider the two jobs J_{i_1}, J_{i_2} with the smallest values of E_i. Among the two left-shifted partial schedules corresponding to sequences (J_{i_1}, J_{i_2}) and (J_{i_2}, J_{i_1}), choose the one with the greatest payoff as the current schedule.

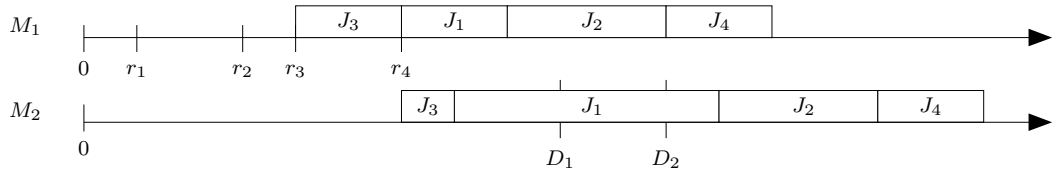


Figure 5.3: The schedule obtained with ECT heuristic for the instance of Table 5.1, of payoff 2.

3. Repeat until all jobs are scheduled: Let J_e be the unscheduled job with the smallest value of E_i. Update the current schedule, by inserting J_e, in the position that maximizes the payoff, while keeping a left-shifted partial schedule.

For the example of Table 5.1, we first compare the two sequences (J_1, J_3) and (J_3, J_1) (see Figure 5.4). (J_1, J_3) is chosen, since it has the greatest payoff. Among the remaining jobs, J_2 has the smallest value of E_i. Thus, we attempt to insert J_2 in all the possible positions, and compare the payoffs (see Figure 5.5).

The sequence providing the best payoff is (J_1, J_3, J_2). Starting from this sequence, in the last step we compare the different positions for inserting J_4 (see Figure 5.6).

Finally, the best schedule of the last step is the final solution. On the example, it is the schedule corresponding to the sequence (J_1, J_3, J_4, J_2).

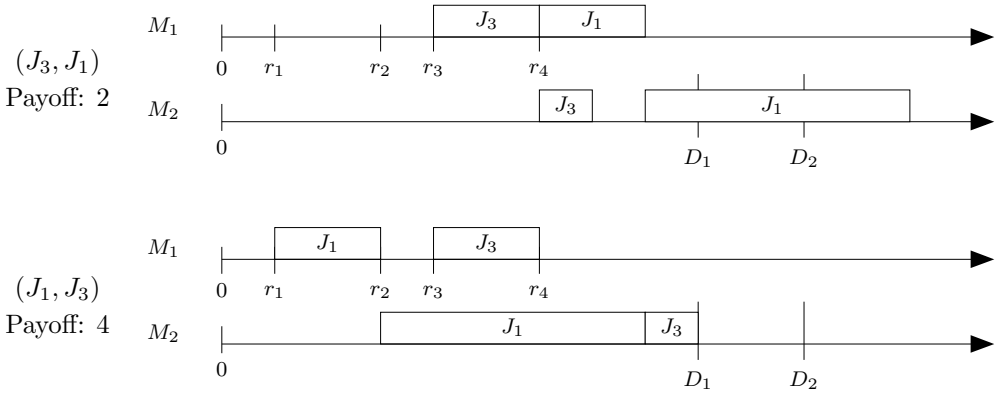


Figure 5.4: First step of NEH heuristic for the instance of Table 5.1.

Step 3 concentrates the largest computational effort of the algorithm. For each tentative position, the method evaluates the solution cost in $O(Nm)$ steps. As each job can be inserted in $O(N)$ positions, the complexity of NEH heuristic for the problem under consideration is $O\big(N^3m\big)$.

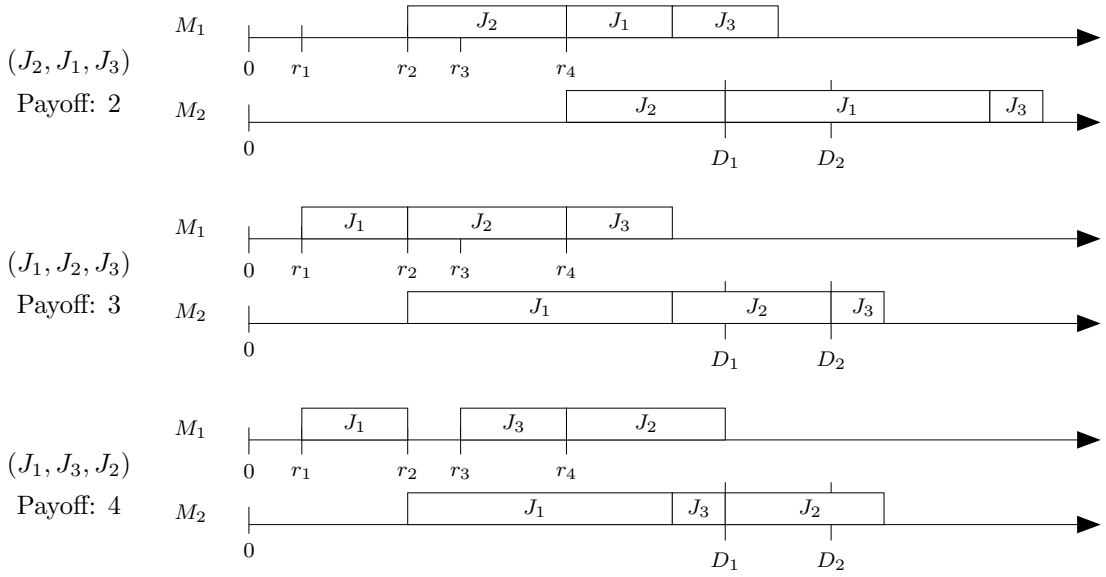


Figure 5.5: Insertions of J_2 at different positions.

The NEH heuristic, originally proposed by Nawaz et al. [22] for a flowshop problem without release dates, was modified by Ladhari and Rakrouki [19] by considering the sorting criterion described above. The method described in [19] is easily adapted for our problem by simply changing the payoff evaluation.

IECT Following the ideas of NEH and ECT methods, we developed a new iterative constructive heuristic. The initial partial schedule is left-shifted and schedules the job with the smallest value of E_i. In the following iterations, the value of E_i is reevaluated for each unscheduled job. Indeed, the earliest possible completion time of a job must now take into account the already scheduled jobs. Then, the job with the smallest E_i is inserted at the end of the current partial schedule. Algorithm 13 describes this $O\big(N^2m\big)$ algorithm.

Let us give an example on the instance of Table 5.1. The first step yields a left-shifted partial schedule scheduling job J_3, since it is the job with the smallest value of E_i (see Figure 5.7). Then, the value of E_i is reevaluated for each of the remaining jobs, as shown in Table 5.2, by taking into account the completion times of job J_3 on both machines. The job with the smallest value of E_{i2} (here, J_4) is inserted at the end of the partial schedule (see Figure 5.8). Afterwards, the values of E_1 and E_2 are reevaluated, as shown in Table 5.3. Since $E_2 < E_1$, the final sequence is (J_3, J_4, J_2, J_1) (see Figure 5.9).

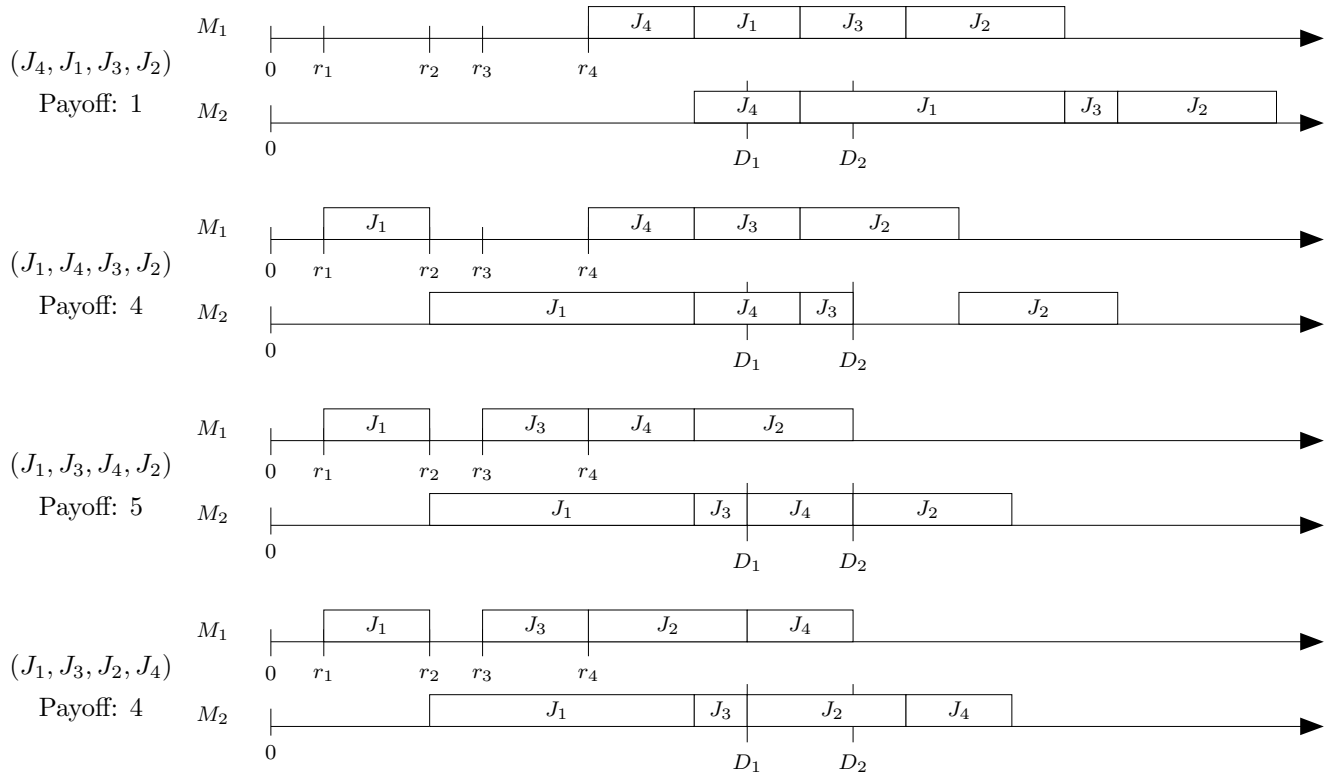


Figure 5.6: Insertions of J_4 at different positions.

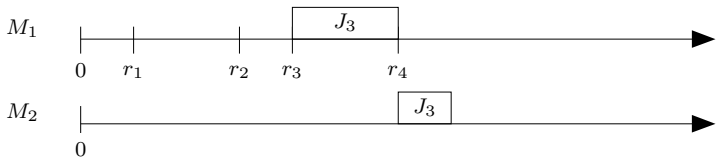


Figure 5.7: First step of IECT heuristics, for the example of Table 5.1.

	$E_{i1} = \max\{r_i, C_{3,1} + p_{i1}\}$	$E_{i2} = \max\{C_{3,2}, E_{i1}\} + p_{i2}$
J_1	$6 + 2 = 8$	$8 + 5 = 13$
J_2	$6 + 3 = 9$	$9 + 3 = 12$
J_4	$6 + 2 = 8$	$8 + 2 = 10$

Table 5.2: The reevaluation of E_i (E_{i2}) for the unscheduled jobs.

Experimental results for the constructive methods

The results are summarized according to K firstly, then to A and finally to R, by giving the average value and the standard deviation, for both *Gap* and *Time*. 100

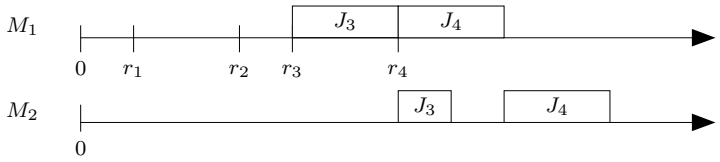


Figure 5.8: Insertion of J_4 in the partial schedule.

	$E_{i1} = \max\{r_i, C_{4,1} + p_{i1}\}$	$E_{i2} = \max\{C_{4,2}, E_{i1}\} + p_{i2}$
J_1	$8 + 2 = 10$	$10 + 5 = 15$
J_2	$8 + 3 = 11$	$11 + 3 = 14$

Table 5.3: The reevaluation of E_i (E_{i2}) for the unscheduled jobs.

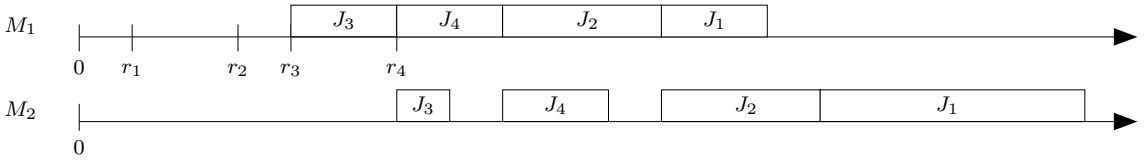


Figure 5.9: The schedule returned by IECT heuristic.

instances are considered for each value of K, and for each value of A. 80 instances are considered for each value of R. Moreover, in the table displaying the gaps related to K, the mean gap on all instances is given.

Tables 5.4 to 5.9 display summaries of the results obtained with the constructive methods. We see that NEH gets the best gaps in all the tables, except when $A = 0.5$, where IECT is the best. Moreover, IECT is the method that gives the best results, after NEH.

We therefore chose to keep both NEH and IECT to be used in the local search methods described in Section 5.4.

We also notice that, similarly to the single machine problem (see the results of the Branch and Bound method, Section 3.5), the instances with a small value of A seem harder to solve.

5.4 Local Search Methods

Local search methods attempt to find cost-improving solutions by exploring the neighborhood of an initial solution. The characterization of a solution neighborhood is arbitrary, and is the element that mainly defines a local search method. If none of the neighbors of a solution has a greater payoff, then the search returns the initial solution as a local

$S \leftarrow \emptyset$;
$\mathcal{J} \leftarrow \mathcal{J}^{all}$;
forall the $J_i \in \mathcal{J}$ **do**
  $E_i = r_i + \sum_{l=1}^{m} p_{il}$;
end
$E_{i^*} \leftarrow \min\{E_i : J_i \in \mathcal{J}\}$;
$S \leftarrow S.J_{i^*}$;
$\mathcal{J} \leftarrow \mathcal{J} \setminus \{J_{i^*}\}$;
while $\mathcal{J} \neq \emptyset$ **do**
  forall the $J_i \in \mathcal{J}$ **do**
    $E_{i1} \leftarrow max\{r_i, C_{i^*,1}\} + p_{i1}$;
    for $l = 2, \ldots, m$ **do**
      $E_{il} \leftarrow max\{C_{i^*,l}, E_{i,l-1}\} + p_{il}$;
    end
  end
  $E_{i^*m} \leftarrow \min\{E_{im} : J_i \in \mathcal{J}\}$;
  $S \leftarrow S.J_{i^*}$;
  $\mathcal{J} \leftarrow \mathcal{J} \setminus \{J_{i^*}\}$;
end

Algorithm 13: Constructive Heuristic IECT.

	ERDH		NEH		ECT		IECT	
K	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
1	14,03%	14,12%	10,22%	10,91%	13,01%	13,02%	11,64%	7,52%
2	14,82%	13,39%	10,25%	9,86%	13,83%	12,65%	12,30%	7,75%
3	15,27%	13,28%	9,79%	9,12%	14,21%	12,37%	12,92%	7,13%
5	15,96%	13,61%	10,08%	9,21%	15,19%	13,24%	14,04%	7,83%
all	15,02%		10,08%		14,06%		12,72%	

Table 5.4: Summary of constructive methods (Gap related to K).

	ERDH		NEH		ECT		IECT	
A	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
0,5	35,78%	4,33%	24,68%	5,20%	33,38%	5,34%	21,78%	5,22%
0,8	16,34%	3,81%	10,74%	3,30%	15,26%	4,09%	14,55%	3,64%
1	6,49%	3,16%	4,16%	2,60%	6,13%	3,13%	10,72%	3,53%
1,2	1,48%	1,69%	0,76%	0,96%	1,48%	1,81%	3,85%	3,03%

Table 5.5: Summary of constructive methods (Gap related to A).

minimum. Otherwise, if there exists a better solution in the neighborhood, a new initial solution is chosen in the neighborhood, and the procedure repeats itself. There are at

	ERDH		NEH		ECT		IECT	
R	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
0,1	14,86%	13,81%	7,50%	8,31%	11,59%	11,21%	9,36%	6,49%
0,3	14,73%	14,19%	9,60%	9,78%	13,40%	13,08%	11,02%	7,28%
0,5	15,26%	13,76%	10,26%	9,82%	14,85%	13,40%	12,75%	7,69%
0,7	15,19%	13,61%	11,34%	10,40%	15,23%	13,46%	14,72%	7,31%
1	15,07%	12,79%	11,73%	10,05%	15,23%	12,64%	15,77%	7,43%

Table 5.6: Summary of constructive methods (Gap related to R).

	ERDH		NEH		ECT		IECT	
K	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
1	0.00	0.00	0.59	0.05	0.00	0.00	0.00	0.00
2	0.00	0.00	0.64	0.03	0.00	0.00	0.00	0.00
3	0.00	0.00	0.66	0.03	0.00	0.00	0.00	0.00
5	0.00	0.00	0.67	0.03	0.00	0.00	0.00	0.00

Table 5.7: Summary of constructive methods (Time related to K).

	ERDH		NEH		ECT		IECT	
A	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
0.5	0.00	0.00	0.60	0.03	0.00	0.00	0.00	0.00
0.8	0.00	0.00	0.64	0.04	0.00	0.00	0.00	0.00
1	0.00	0.00	0.65	0.05	0.00	0.00	0.00	0.00
1.2	0.00	0.00	0.67	0.05	0.00	0.00	0.00	0.00

Table 5.8: Summary of constructive methods (Time related to A).

	ERDH		NEH		ECT		IECT	
R	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
0.1	0.00	0.00	0.61	0.06	0.00	0.00	0.00	0.00
0.3	0.00	0.00	0.62	0.05	0.00	0.00	0.00	0.00
0.5	0.00	0.00	0.64	0.04	0.00	0.00	0.00	0.00
0.7	0.00	0.00	0.65	0.04	0.00	0.00	0.00	0.00
1	0.00	0.00	0.67	0.04	0.00	0.00	0.00	0.00

Table 5.9: Summary of constructive methods (Time related to R).

least two ways of selecting the new initial solution. The *best improvement* strategy [31] chooses a neighbor with the best cost (or payoff) of the whole neighborhood. The *first improvement* strategy [31] chooses the first found neighbor that has a better cost (or payoff) than the initial solution.

5.4.1 Neighborhoods

The two neighborhoods explored in this work are based on the standard ones studied by den Besten and Stützle [1] for scheduling problems. Recall that any feasible left-shifted schedule of $F|r_j, perm|\sum_{k=1}^{K} V_k$ can be represented as a sequence.

- In the *interchange* neighborhood, a neighbor sequence is obtained by swapping the two jobs at positions j and j' of the initial schedule, $j \in \{1,\ldots,N\}$, $j' \in \{1,\ldots,N\}\backslash\{j\}$. The size of this neighborhood is thus $N(N-1)/2$.
- The *insert* neighborhood constructs sequences where the job that is in position j, $j \in \{1,\ldots,N\}$, in the initial schedule is moved to a new position j', $j' \in \{1,\ldots,N\}\backslash\{j\}$. Hence, the jobs that were in some position $j'' \in \{j+1,\ldots,j'\}$ are now at position $j''-1$ in the constructed sequence, while the positions of the other jobs remain unchanged. To avoid identical neighbors, the first job can be moved to $N-1$ positions, while the other jobs can only be moved to $N-2$ different positions. Hence, the whole neighborhood of a sequence contains $(N-1)^2$ sequences.

For both methods, the cost of each neighbor is computed in $\Theta(Nm)$ steps, which makes the local search running in $O\big(N^3m\big)$.

Experimental results on local search

The results are summarized according to K firstly, then to A and finally to R, by giving the average value and the standard deviation, for both *Gap* and *Time*. 100 instances are considered for each value of K, and for each value of A. 80 instances are considered for each value of R. Moreover, in the table displaying the gaps related to K, the mean gap on all instances is given.

The experiments described in this section aim to evaluate the impact of the local search methods on the solutions obtained by the constructive methods. As said at the end of Section 5.3, we evaluate the local search on the solutions obtained by both constructive methods NEH and IECT. *Interchange (Itc)* and *Insertion (Ist)* neighborhoods were implemented by following the *first improvement (fi)* and the *best improvement (bi)* strategies. Tables 5.10 to 5.15 show the results for NEH, while Tables 5.16 to 5.21 show the results for IECT.

The solutions constructed with NEH algorithm were only slightly improved by the local search. Moreover, starting from NEH solution, both neighborhoods (*Ist* and *Ict*) and both improvement strategies (*best* and *first*) give similar results. As expected, the *best improvement* strategy needs more computation time than the *first improvement* one.

Visible improvements of the IECT solutions are, instead, produced by the local search, especially with *Ist* neighborhood.

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
K	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
1	9,54%	10,32%	9,54%	10,32%	9,43%	10,45%	9,45%	10,48%
2	9,57%	9,35%	9,58%	9,35%	9,44%	9,39%	9,45%	9,38%
3	9,15%	8,53%	9,19%	8,61%	9,31%	8,82%	9,31%	8,80%
5	9,37%	8,42%	9,33%	8,39%	9,57%	8,85%	9,57%	8,86%
all	9,41%		9,41%		9,44%		9,44%	

Table 5.10: Summary of NEH local search (Gap related to K).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
A	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
0,5	23,00%	5,13%	23,01%	5,17%	23,50%	5,01%	23,52%	5,03%
0,8	10,09%	3,05%	10,10%	3,05%	10,01%	3,07%	10,02%	3,06%
1	3,85%	2,55%	3,85%	2,55%	3,57%	2,30%	3,58%	2,30%
1,2	0,70%	0,89%	0,69%	0,89%	0,66%	0,88%	0,66%	0,88%

Table 5.11: Summary of NEH local search (Gap related to A).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
R	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
0,1	7,12%	7,88%	7,09%	7,83%	7,31%	8,13%	7,31%	8,13%
0,3	8,90%	9,40%	8,88%	9,38%	8,94%	9,47%	8,94%	9,47%
0,5	9,53%	9,25%	9,51%	9,21%	9,69%	9,56%	9,70%	9,55%
0,7	10,31%	9,30%	10,38%	9,41%	10,30%	9,79%	10,35%	9,83%
1	11,18%	9,55%	11,20%	9,57%	10,94%	9,60%	10,93%	9,58%

Table 5.12: Summary of NEH local search (Gap related to R).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
K	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
1	0.97	0.20	1.05	0.32	5.18	2.20	4.42	2.34
2	1.10	0.26	1.17	0.37	5.72	2.77	5.07	3.38
3	1.14	0.25	1.22	0.34	5.29	2.35	4.68	2.95
5	1.26	0.36	1.45	0.64	6.13	2.63	5.53	3.50

Table 5.13: Summary of NEH local search (Time related to K).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
A	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
0.5	1.26	0.42	1.53	0.70	6.31	2.62	5.99	3.89
0.8	1.16	0.30	1.23	0.38	5.90	2.68	5.19	3.02
1	1.06	0.17	1.11	0.22	5.80	2.90	5.01	3.04
1.2	0.99	0.10	1.02	0.16	4.31	0.88	3.51	1.37

Table 5.14: Summary of NEH local search (Time related to A).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
R	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
0.1	1.02	0.26	1.11	0.45	4.50	1.40	3.72	2.27
0.3	1.12	0.29	1.23	0.41	5.63	2.22	5.04	2.71
0.5	1.14	0.32	1.28	0.58	5.44	2.36	4.73	3.00
0.7	1.19	0.33	1.30	0.49	6.35	3.17	5.77	3.70
1	1.12	0.22	1.19	0.32	5.97	2.75	5.37	3.26

Table 5.15: Summary of NEH local search (Time related to R).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
K	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
1	9,30%	7,36%	9,28%	7,27%	6,66%	5,49%	6,64%	5,51%
2	10,22%	7,33%	10,30%	7,29%	5,66%	5,18%	5,76%	5,15%
3	11,07%	7,06%	11,07%	7,05%	5,55%	4,89%	5,76%	4,78%
5	12,35%	7,66%	12,38%	7,68%	6,07%	5,82%	6,11%	5,52%
all	10,73%		10,76%		5,99%		6,07%	

Table 5.16: Summary of IECT local search (Gap related to K).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
A	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
0,5	19,34%	5,60%	19,36%	5,56%	12,77%	4,88%	12,63%	4,73%
0,8	12,41%	4,05%	12,40%	4,03%	6,11%	3,25%	6,28%	3,30%
1	8,50%	4,27%	8,57%	4,19%	3,91%	2,28%	4,15%	2,14%
1,2	2,68%	2,73%	2,71%	2,72%	1,15%	1,25%	1,21%	1,29%

Table 5.17: Summary of IECT local search (Gap related to A).

5.4.2 Variable Neighborhood Descent

Neighborhoods can be combined, in order to better explore the solutions space. Variable Neighborhood Descent (VND) [14] establishes an order in which the local search methods will be applied. Starting from a solution generated by a constructive algorithm, and

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
R	Gap	Std dev	Gap	Std dev	Gap	Std dev	Gap	Std dev
0,1	7,09%	5,98%	7,19%	6,06%	4,91%	4,74%	4,84%	4,48%
0,3	8,67%	6,84%	8,78%	6,81%	5,42%	5,57%	5,49%	5,38%
0,5	10,43%	7,19%	10,39%	7,19%	5,93%	5,42%	5,89%	5,32%
0,7	12,67%	7,31%	12,64%	7,23%	6,52%	5,49%	6,85%	5,48%
1	14,80%	7,18%	14,79%	7,17%	7,15%	5,36%	7,28%	5,23%

Table 5.18: Summary of IECT local search (Gap related to R).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
K	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
1	0.47	0.17	0.84	0.44	10.35	4.99	11.91	5.86
2	0.67	0.26	1.02	0.38	15.65	7.03	17.09	7.85
3	0.76	0.35	1.22	0.50	19.58	7.91	19.58	6.93
5	1.03	0.39	1.60	0.54	26.40	10.03	23.47	7.16

Table 5.19: Summary of IECT local search (Time related to K).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
A	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
0.5	0.64	0.26	1.11	0.44	15.45	5.49	18.50	6.21
0.8	0.77	0.31	1.28	0.49	22.12	9.32	21.68	7.16
1	0.82	0.35	1.30	0.53	22.35	10.47	20.48	7.64
1.2	0.69	0.47	0.99	0.64	12.05	8.43	11.39	7.35

Table 5.20: Summary of IECT local search (Time related to A).

	Itc-fi		Itc-bi		Ist-fi		Ist-bi	
R	Time	Std dev	Time	Std dev	Time	Std dev	Time	Std dev
0.1	0.74	0.36	1.19	0.52	12.38	7.41	13.31	6.96
0.3	0.77	0.37	1.27	0.55	15.47	8.56	15.57	7.64
0.5	0.80	0.38	1.33	0.53	18.35	9.05	18.44	7.55
0.7	0.76	0.43	1.29	0.56	22.05	10.62	20.89	7.68
1	0.58	0.21	0.78	0.36	21.72	8.90	21.85	7.69

Table 5.21: Summary of IECT local search (Time related to R).

a sequence of q neighborhoods, a VND exploits the first neighborhood until a local optimum is found. This solution becomes the initial solution to the next neighborhood, and so on until the q-th neighborhood. If the solution returned by the q-th local search is better than the initial solution at the beginning of the first local search, the procedure restarts again from the first neighborhood. Otherwise, the search stops, and the returned

solution is the solution returned by the q-th local search.

The pseudocode of this method is given below. We suppose that $LOC\_SEARCH(\mathcal{N}, S)$ is a function returning the solution found by a local search with neighborhood $\mathcal{N}$ starting from solution S.

Input: $\mathcal{N}_1, \ldots, \mathcal{N}_n$, S
Output: S^{best}
1 $S^{best} \leftarrow S$
2 $S' \leftarrow S$
3 flag $\leftarrow$ true
4 **while** *flag* **do**
5 **for** $i = 1$ ***to*** n **do**
6 $S' \leftarrow LOC\_SEARCH(\mathcal{N}_i, S')$
7 **if** $\mathcal{F}(S') = \mathcal{F}(S^{best})$ **then**
8 flag $\leftarrow$ false
9 **else**
10 $S^{best} \leftarrow S'$
11 **return** S^{best}

Algorithm 14: Variable Neighborhood Descent.

Experimental results on variable neighborhood descent

The results are summarized according to K firstly, then to A and finally to R, by giving the average value and the standard deviation, for both *Gap* and *Time*. 100 instances are considered for each value of K, and for each value of A. 80 instances are considered for each value of R. Moreover, in the table displaying the gaps related to K, the mean gap on all instances is given.

In the next experiment, we combined the local searches *Itc-bi* and *Ist-bi* in a VND method, so that two VND variants were created by changing the order of the interchange and insertion local searches. We only applied VND on the solutions obtained by the IECT constructive, since we saw in the previous experiments that local search has more impact on the IECT solutions. The results are displayed in Tables 5.22 to 5.27.

The two VND methods are comparable regarding gaps. However, *Itc-bi + Ist-bi* is faster.

Besides, by comparing these results with those of local search, we note there is not an impressive improvement in solutions quality. Similar conclusions were presented by den Besten and Stützle [1] about the VND application to the permutation flowshop problem with the objective of minimizing makespan.

	Itc-bi+Ist-bi		Ist-bi+Itc-bi	
K	Gap	Std dev	Gap	Std dev
1	6,27%	5,56%	6,24%	5,62%
2	5,63%	5,21%	5,56%	5,13%
3	5,58%	4,61%	5,59%	4,67%
5	6,08%	5,42%	5,97%	5,48%
all	5,89%		5,84%	

Table 5.22: Summary of IECT-VND methods (Gap related to K).

	Itc-bi+Ist-bi		Ist-bi+Itc-bi	
A	Gap	Std dev	Gap	Std dev
0,5	12,44%	4,46%	12,40%	4,67%
0,8	6,20%	3,34%	6,09%	3,33%
1	3,73%	2,42%	3,72%	2,28%
1,2	1,18%	1,28%	1,15%	1,30%

Table 5.23: Summary of IECT-VND methods (Gap related to A).

	Itc-bi+Ist-bi		Ist-bi+Itc-bi	
R	Gap	Std dev	Gap	Std dev
0,1	12,09%	4,22%	11,99%	4,39%
0,3	8,11%	5,21%	8,15%	5,34%
0,5	4,89%	3,06%	4,79%	2,94%
0,7	2,89%	2,59%	2,83%	2,48%
1	1,46%	1,28%	1,44%	1,30%

Table 5.24: Summary of IECT-VND methods (Gap related to R).

	Itc-bi+Ist-bi		Ist-bi+Itc-bi	
K	Time	Std dev	Time	Std dev
1	11.31	5.68	14.70	6.43
2	16.70	7.27	20.01	7.90
3	20.01	7.28	22.65	6.72
5	24.21	8.70	26.53	7.35

Table 5.25: Summary of IECT-VND methods (Time related to K).

5.5 GRASP heuristic

In order to attempt to improve the previous results, particularly the gap, we considered the use of the following metaheuristic. A *Greedy Randomized Adaptive Search Procedure* (GRASP) is a multi-start metaheuristic which consists of applying local search to feasible

	Itc-bi+Ist-bi		Ist-bi+Itc-bi	
A	Time	Std dev	Time	Std dev
0.5	19.39	7.48	21.48	6.04
0.8	21.34	7.28	24.72	7.18
1	20.66	9.01	23.75	7.55
1.2	10.84	6.43	13.94	7.78

Table 5.26: Summary of IECT-VND methods (Time related to A).

	Itc-bi+Ist-bi		Ist-bi+Itc-bi	
R	Time	Std dev	Time	Std dev
0.1	13.01	7.17	16.40	7.48
0.3	15.27	7.82	18.52	7.87
0.5	17.92	7.98	21.34	7.60
0.7	20.96	8.82	23.69	7.60
1	23.14	7.64	24.91	8.01

Table 5.27: Summary of IECT-VND methods (Time related to R).

starting solutions generated with a greedy randomized constructive heuristic. It was introduced by Feo and Resende [9] for solving a set covering problem with unit costs.

In the constructive phase, an iterative procedure builds a solution from scratch, adding one element at a time to a partial solution, until the solution is feasible. The addition of each element is evaluated according to a greedy function that verifies the benefit of including this element in the partial solution. If the function is purely greedy, it chooses the element that represents the greatest benefit. Instead, GRASP uses a randomized greedy procedure which keeps a restricted candidate list (RCL) formed by the best elements. At each iteration, an element is chosen at random from the RCL. After adding the chosen element in the partial solution, the remainder elements must be reevaluated.

Solutions built with the randomized greedy algorithm are not guaranteed to be locally optimal, even with respect to simple neighborhood structures. Therefore, the application of local search to such a solution usually results in an improved locally optimal solution.

Starting from an initial solution, local search explores its neighborhood for a cost-improving solution. If none is found, then the search returns the initial solution as a local minimum. Otherwise, if an improving solution is found, it is made the new initial solution, and the procedure repeats itself.

Algorithm 15 shows the pseudo-code for the GRASP procedure.

The GRASP heuristic developed for $F|r_j, perm|\sum_{k=1}^{K} V_k$ is based on the results

```
1  GRASP(StopCriterion, Seed)
2  z* ← 0;
3  while StopCriterion is not satisfied do
4  |  S ← GreedyRandomizedAlgorithm(Seed);
5  |  S ← LocalSearch(S);
6  |  if F(S) > z* then
7  |  |  S* ← S;
8  |  |  z* ← F(S);
9  |  end
10 end
```

Algorithm 15: Template of a GRASP heuristic for maximization.

presented in the previous sections.

The constructive phase of GRASP is a randomized variant of the IECT greedy algorithm. At each iteration, the jobs not in the solution are still evaluated by the greedy function E_j, as defined above. However, instead of choosing the job with the smallest E_i, the randomized algorithm first identifies the minimum (E^-) and maximum (E^+) greedy function values of the candidate elements. Then, a restricted candidate list (RCL), formed by all candidate elements whose greedy function value is greater than or equal to $E^+ - \alpha(E^+ - E^-)$, is built. A job index j^r is chosen at random from the RCL and the corresponding job J_{j^r} is inserted in the scheduling at the end of the sequence. The chosen value of α is 0.1, since we observed in some preliminary experiments that this value led to the best results.

In the GRASP local search, the method $Ist - bi$ is followed by $Itc - bi$ in a VND strategy. The stopping criterion is a time limit of 30 minutes. For each instance, four initial randomized solutions were generated. After the four corresponding local search iterations, the best solution is kept.

Experimental results for GRASP

The results are summarized according to K firstly, then to A and finally to R, by giving the average value and the standard deviation, for both *Gap* and *Time*. 100 instances are considered for each value of K, and for each value of A. 80 instances are considered for each value of R. Moreover, in the table displaying the gaps related to K, the mean gap on all instances is given.

The experimental results on GRASP heuristic are summarized in Tables 5.28 to 5.30. Unfortunately, GRASP does not improve dramatically the gaps of the obtained solutions.

K	Gap	Std dev
1	5,28%	5,22%
2	5,30%	5,07%
3	5,36%	4,61%
5	5,92%	5,54%
all	5,46%	

Table 5.28: Summary of grasp (Gap related to K).

A	Gap	Std dev
0,5	11,89%	4,61%
0,8	5,56%	3,28%
1	3,32%	2,41%
1,2	1,08%	1,28%

Table 5.29: Summary of grasp (Gap related to A).

R	Gap	Std dev
0,1	11,3%5	4,24%
0,3	7,51%	5,55%
0,5	4,41%	3,15%
0,7	2,70%	2,46%
1	1,35%	1,30%

Table 5.30: Summary of grasp (Gap related to R).

Overall, local search has provided a noticeable improvement in gaps: the best constructive heuristic has a mean gap of 10.08%, while the best local search has a gap of 5.84%. As for GRASP, it only yields little improvement: from 5.84% to 5.46%. However, we tested here one possible configuration for GRASP: deeper experimental analysis would be necessary to find the tunings that would give the best results.

5.6 Conclusion

In this chapter, we studied a permutation flowshop problem, which is strongly NP-hard, as it is a generalization of the single machine problem. We tackled this problem with several heuristic methods: four constructive heuristics, three of them found in the literature, and a new one; and some local search methods, based on the classical Interchange and Insertion neighborhoods (simple local search, variable neighborhood search, and GRASP). The experimental results show that the simple local search and the variable neighborhood search provide good solutions in a reasonable time.

Conclusion and research issues

Starting from a real world digitization workflow issue, we identified a scheduling problem with a new criterion involving common delivery dates for the jobs. In order to focus on the optimization criterion/release dates pair, for characterizing structural properties of the optimal solutions, we considered a single machine problem.

In the studied problem, each job has its own processing time and release date. Additionally, all the jobs share K common delivery dates. The objective is to attain some target quantities (fixed by the client) of digitized books at each delivery date, while maximizing the payoff of the manufacturer. In order to take into account both aspects, the considered objective function aims to maximize the number of jobs processed between time 0 and each delivery date. Hence, we have a cumulative aspect, since the jobs processed before a given delivery date are also processed before the subsequent delivery dates, and may thus be counted several times when computing the total payoff. The same objective function can alternatively be represented as a sum of stepwise payoff functions associated to the jobs. Each job has the same fixed decreasing stepwise function, whose values range from K if the job completes at or before the first delivery date, to 0 for a completion after the last delivery date, and decreases by 1 at each delivery date.

Though problems considering a generalization of this criterion have been studied already, we attempted here to precisely delimit the frontiers of the complexity classes. For this purpose, we established the complexity of the general problem (strongly NP-hard) and of some special cases. Among them, we showed that the problem with fixed K is weakly NP-hard, and that the problem with a single delivery date is polynomial.

For solving the Single Delivery Date problem, we provided a polynomial algorithm, which besides enables the computation of efficient bounds for the general problem. Moreover, we established some dominance rules. With these tools, we have been able to provide a fast Branch and Bound algorithm for the general problem.

We have also considered the weakly-NP hard problem with two delivery dates, for

which a pseudopolynomial algorithm based on dynamic programming has been provided. Moreover, we established a dedicated dominance rule for this problem. Thanks to this dominance rule, we have been able to improve the dynamic programming algorithm. Finally, we showed a polynomial algorithm yielding feasible solutions with an absolute performance guarantee of 1.

Lastly, we presented some heuristic algorithms for the permutation flowshop problem with the same criterion: constructive heuristics, one of which is new, local search methods, and a GRASP algorithm.

All the mentioned algorithms were implemented. The dynamic programming algorithm can solve in a reasonable time instances with up to 60 jobs (less than 12 minutes). With the Branch and Bound algorithm, we tackled instances with up to 20 delivery dates and 2000 jobs: 85% of all the tested instances were optimally solved in less than 2 minutes; while the mean gap over all instances unsolved after 15 minutes is less than 0.5%. Finally, the best heuristic methods for the flowshop problem yield solutions with gaps inferior to 6% in less than 25 seconds, for instances with 100 jobs and up to 5 delivery dates.

As the problem comes from a real world issue, several extensions of the problem can be considered, which take into account different constraints of the real problem that were not considered in this first study. We describe some of these extensions below.

First, it is worth investigating if the polynomial algorithm with approximation guarantee for the two delivery dates case can be extended to the general case. Besides the possibility of rapidly obtaining a good solution for the original problem, it would also enable to tackle the problem of retreating jobs when the quality control is not satisfying, in the industrial workflow. In this case, a good solution must indeed be instantly recalculated in order to reschedule the affected jobs.

Another aspect that was not yet considered is the following: the manufacturer must satisfy the delivery dates according to the number of books, however its payment is proportional to the number of digitized pages. A possible way of tackling this problem is to assign weights, proportional to the number of pages, to each job, and to integrate them in the cumulative objective function. Hence, the payoff of a job J_j with weight w_j is simply w_j times the payoff of a job in the unweighted version of the problem. With the introduction of weights, we lose part of the structure of the original problem, since there is no more symmetry between the jobs, while many results of this work were based on the number of jobs. Moreover, the idea of an adaptation of SDD-algorithm, in order

to have a basic polynomial algorithm for the weighted problem, is not possible. Indeed, the weighted single delivery date problem is equivalent to $1||\sum w_i U_i$ (as the unweighted problem is equivalent to $1||\sum U_i$), which is NP-hard. Hence, it seems unlikely to obtain as good bounds for the weighted problem as for the unweighted problem. The weighted problem is also a particular case of the problem considered by Detienne et al. [7], hence their method is directly applicable to the problem. Their more general formulation can thus be a possible starting point for a dedicated exact method for the weighted problem.

However, we must also point out that aggregating weights with the delivery dates objective function can give unbalanced solutions, i.e. where one among the client and the manufacturer is poorly satisfied. For instance, it could happen that at a given delivery date, a few jobs with large weights are scheduled: in this case the number of processed jobs can be quite far from the target quantity of jobs. A natural answer to this issue is to introduce a multicriteria aspect (bicriteria here), where the balance of the solution can be explicitly controlled by choosing good compromise solutions among the Pareto optima. In this case, the payoff of a job would have two components: the first one identical to the payoff of the unweighted problem, while the second one would be a function of its weight.

Furthermore, another possible aspect to consider, related to the industrial issue, is the fairness aspect: the manufacturer has several clients, and must attempt to serve them in such a way that they are all rather satisfied, if possible. This would introduce non-linear aggregation functions on the satisfaction-based utilities of the clients, like for instance OWA [34].

Finally, the study of the flowshop problem with delivery dates and cumulative payoffs is certainly relevant, since it reflects the real structure of the workflow. The current work on the permutation flowshop needs to be enhanced by attempting to identify structural properties, and especially designing good upper bounds (for more than two machines) in order to evaluate the heuristic methods. Moreover, in addition to the already presented heuristic methods, other metaheuristics, such as Iterated Local Search [20], seem worth considering.

Afterwards, the next step could be to consider a flowshop problem that is not a permutation flowshop.

Additionally, by modeling the problem as a flowshop (for instance with four machines, as there are four main steps in the digitization workflow), the following specification of the real problem can be taken into account. Each book that is to be digitized must be sent back to the client four weeks after receiving it, even if the whole process is not completed. This means that there is an intermediate due date for each job for completing the two first main steps (Digitization and Preliminary quality control), and if this due

date is not satisfied, there is a damage for the client, as its demand is not fulfilled (i.e. having the book back at the due date). Hence, besides the cumulative objective function, we can consider the intermediate due dates as hard constraints, or alternatively as soft constraints, which generate penalties if they are not satisfied (it seems more likely that some due date might not be satisfied, due to possibly several reprocessings).

Bibliography

[1] den Besten, M. and Stützle, T. (2001). Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling. *Proceedings of The Fourth Metaheuristics International Conference*, 545–549.

[2] Brucker, P. (2007). *Scheduling Algorithms*. Berlin: Springer.

[3] Carlier, J. and Chrétienne, P. (1988). *Problèmes d'ordonnancement: modélisation, complexité, algorithmes*. Masson.

[4] Curry, J. and Peters, B. (2005). Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stabimity objectives. *International Journal of Production Research, 43(15)*, 3231–3246.

[5] Della Croce, F., Gupta, J.N.D. and Tadei, R. (2000). Minimizing tardy jobs in a flowshop with common due date. *European Journal of Operational Research, 120(2)*, 375–381.

[6] Della Croce, F., Grosso, A. and Salassa, F. (2011). A Matheuristic Approach for the Total Completion Time Two-Machines Permutation Flow Shop Problem. In *Evolutionary Computation in Combinatorial Optimization, LNCS*, 38–47.

[7] Detienne, B., Dauzères-Pérès, S. and Yugma, C. (2011). Scheduling jobs on parallel machines to minimize a regular step total cost function. *Journal of Scheduling, 14(6)*, 523–538.

[8] Detienne, B., Dauzères-Pérès, S., and Yugma, C. (2012). An exact approach for scheduling jobs with regular step cost functions on a single machine. *Computers & Operations Research, 39(5)*, 1033–1043.

[9] Feo, T.A. and Resende, M.G.C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters, 8(2)*, 67–71.

[10] Garey, M. R., Johnson, D. S. and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1(2)*, 117–129.

[11] Graham, R. L., Lawler, E. L., Lenstra J. K. and Rinnooy Kann A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326 .

[12] Hall,N. G., Lesaoana M. and Potts C. N. (2001). Scheduling with Fixed Delivery Dates. *Operations Research, 49(1)*, 134–144 .

[13] Hall, N. G., Sethi, S. P. and Sriskandarajah,C. (1991). On the complexity of generalized due date scheduling problems. *European Journal of Operational Research, 51(1)*, 100–109.

[14] Hansen, P. and Mladenovic, N. (2003). Variable Neighborhood Search. *Handbook of Metaheuristics*. New York: Springer.

[15] Janiak, A. and Krysiak, T. (2007). Single processor scheduling with job values depending on their completion times. *Journal of Scheduling, 10(2)*, 129–138.

[16] Janiak, A. and Krysiak, T. (2012). Scheduling jobs with values dependent on their completion times. *International Journal of Production Economics, 135(1)*, 231–241.

[17] Kellerer, H., Pferschy, U. and Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer

[18] Kise, H., Ibaraki, T., and Mine, H. (1978). A solvable case of the one-machine scheduling problem with ready and due times. *Operations Research, 26(1)*, 121–126.

[19] Ladhari, T. and Rakrouki, M. A. (2009). Heuristics and lower bounds for minimizing the total completion time in a two-machine flowshop. *International Journal of Production Economics, 122(2)*, 678–691

[20] Lourenco, H., Martin, O. and Stützle, T. (2003). Iterated Local Search. *Handbook of Metaheuristics*. New York: Springer.

[21] Moore, J. M. (1968). An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs. *Management Science, 15(1)*, 102–109.

[22] Nawaz, M., Enscore, E. E. and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11(1)*, 91–95.

[23] Pinedo, M. (1995). *Scheduling Theory, Algorithms, and Systems*. Englewood Cliffs, New Jersey: Prentice Hall.

[24] Potts, C.N. (1985). Analysis of heuristics for two-machine flow-shop sequencing subject to release dates. *Mathematics of Operations Research, 10(4)*, 576–584.

[25] Rakrouki, M. A. and Ladhari, T. (2009). A branch-and-bound algorithm for minimizing the total completion time in two-machine flowshop problem subject to release dates. *Proceedings of International Conference on Computers & Industrial Engineering*, 80–85.

[26] Raut, S., Swami, S. and Gupta, J. N. D. (2008). Scheduling a capacitated single machine with time deteriorating job values. *International Journal of Production Economics, 114(2)*, 769–780.

[27] Sahin, G. and Ahuja, R. K. (2011). Single-machine scheduling with stepwise tardiness costs and release times. *Journal of Industrial Management and Optimization, 7(4)*, 825–848.

[28] Seddik, Y., Gonzales, C. and Kedad-Sidhoum, S. (2011). Single machine scheduling with delivery dates and cumulative payoffs. *Proceedings of the 5th Multidisciplinary Scheduling Conference (MISTA)*, 261–274.

[29] Seddik, Y., Gonzales, C. and Kedad-Sidhoum, S. (2011). Solving the one-machine scheduling problem with cumulative payoffs. *10th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2011)*.

[30] Seddik, Y., Gonzales, C. and Kedad-Sidhoum, S. (2012). A Branch and Bound method for a one-machine scheduling problem with cumulative payoffs. *International Symposium on Combinatorial Optimization (CO 2012)*.

[31] Talbi, E.-G. (2009). *Metaheuristics - From Design to Implementation*. Wiley.

[32] Tseng, C.-T., Chou, Y.-C. and Chen, W.-Y. (2010). A variable neighborhood search for the single machine total stepwise tardiness problem. *Proceedings of the 2010 International Conference on Engineering, Project and Production Management*, 101–108.

[33] Vallada, E., Ruiz, R. and Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research, 35(4)*, 769–780.

[34] Yager, R. (1988). On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man and Cybernetics, 18*, 183–190.

[35] Yang, W.-H. (2009). Sequencing jobs subject to multiple common due dates. *Proceedings of the 10th Asia Pacific Industrial Engineering & Management Systems Conference*, 1041–1050.

Résumé

Le problème étudié dans cette thèse est issu d'une problématique réelle, concernant l'optimisation du processus de numérisation des ouvrages de la Bibliothèque Nationale de France (BNF). La modélisation de ce problème met en évidence un critère d'optimisation nouveau en ordonnancement, tenant compte de gains cumulatifs liés à des dates de livraison communes à toutes les tâches. Dans le but d'identifier les structures des solutions optimales liées à ce nouveau critère et à des dates de disponibilité des tâches, nous nous sommes surtout concentrés sur un problème d'ordonnancement à une machine. Nous avons identifié les classes de complexité de ce problème, et proposé une méthode de résolution exacte de type Branch and Bound pour le problème général, s'appuyant sur des bornes et des règles de dominance dédiées. Nous avons également considéré le problème à deux dates de livraison (NP-difficile au sens faible), pour lequel nous avons proposé un algorithme pseudopolynomial de programmation dynamique et un algorithme d'approximation polynomial avec une performance de garantie absolue égale à 1. Enfin, dans le but de nous rapprocher de la problématique industrielle, nous nous sommes intéressés à un problème de flowshop de permutation, avec le même critère d'optimisation. Pour ce problème, nous avons proposé plusieurs heuristiques : des algorithmes constructifs, des algorithmes de recherche locale, et une métaheuristique de type GRASP. Tous les algorithmes ont été implémentés, en particulier le Branch and Bound pour le problème à une machine et la recherche locale pour le flowshop permettent d'obtenir de bonnes solutions en temps raisonnable.

Mots-clés : Ordonnancement, dates de livraison, dates de disponibilité, gains cumulatifs, fonctions de gain en escalier, une machine, flowshop.

Abstract

Starting from a real world digitization workflow issue, we identified a scheduling problem with a new criterion involving common delivery dates for the jobs. In order to focus on this new criterion and on the jobs' release dates, we mainly worked on a single machine problem. We delimited the complexity classes of the problem, and provided a Branch and Bound algorithm for the general problem, based on dedicated bounds and dominance rules. We also considered the weakly NP-hard problem with two delivery dates, for which we designed a pseudopolynomial dynamic programming algorithm and an approximation algorithm with an absolute performance guarantee of 1. Finally, in order to consider a problem more closely related to the industrial issue, we studied a permutation flowshop problem with the same criterion. For this problem, we proposed several heuristic methods: constructive algorithms, local search, and a GRASP algorithm. All the algorithms were implemented. In particular the Branch and Bound method for the single machine problem and the local search algorithms for the flowshop provide good solutions in a reasonable time.

Keywords: Scheduling, delivery dates, release dates, cumulatives payoffs, stepwise payoff functions, single machine, flowshop.